UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Carrera de Ingeniería de Sistemas

Resolución de un problema de selección de áreas óptimas para forestación, representado como un «Knapsack Problem», mediante la aplicación de un Algoritmo Genético y técnicas de Topología.

Trabajo de titulación previo a la obtención
del título de Ingeniero en Sistemas

Autor:

Luis Eduardo Lazo Vera
CI: 0301968186

Director:

Ing. Lizandro Damián Solano Quinde, PhD
CI: 0102428893

CUENCA—ECUADOR
2018

Resumen

Iniciativas gubernamentales y asociaciones de protección al medio ambiente, a menudo, están interesadas en proyectos de forestación con el objetivo de reducir la producción de sedimento que converge en los ríos. Seleccionar áreas óptimas donde se plantará bosque para minimizar la erosión del suelo, sujeto a restricción presupuestaria representa un reto de optimización. Áreas potenciales donde se plantará bosque, se digitalizan en mapas ráster, donde se almacena beneficios de forestación y costo asociados de cada celda. En función de la resolución que se use, el número de celdas resultante, podría llegar a ser impráctico para algoritmos de optimización comunes. En este manuscrito, se aplica el algoritmo *CAMF (Cellular Automata Based Heuristic Solution Method for Minimizing Flow, CAMF)* para construir un ranking de celdas candidatas, con su beneficio asociado de forestación (reducción de sedimento) y costo de forestación (distancia a vías). Al tener dos objetivos concurrentes (minimizar la producción de sedimento y presupuesto limitado), permite, modelar el problema como un «Knapsack Problem», y se propone variantes del Algoritmo Genético con técnicas de topología como heurística, para su resolución, los valores obtenidos se contrastan con el valor del óptimo global calculado con «Branch and Bound». Los resultados obtenidos con «Branch and Bound» muestran ser superiores a estrategias como *Shani's PTAS algoritmo* integrado a *CAMF-B*. «Branch and Bound» es aplicable a problemas de forestación donde intervienen miles de celdas. En situaciones de proyectos con excesivo números de celdas donde no es posible aplicar «Branch and Bound», debido al costo computacional, el Algoritmo Genético siempre encuentra una solución factible con costo-beneficio cercanos al óptimo global.

Palabras clave: *CAMF, CAMF-B, Forestación, Knapsack problem, Algoritmo genético, Branch and Bound, Topología*

Abstract

Policy initiatives and decision makers dealing with environmental conservation are interested in forestation projects in order to minimize runoff sediment reaching riverbeds. Selecting the optimal areas in which to plant trees to minimize soil erosion, while remaining within budgetary constraints, is a difficult optimization challenge. Potential reforestation terrain can be digitized in raster maps where erosion and cost of planting can be represented for each cell. Due to the resolution used in such raster maps, the number of resultant cells can be unworkable for common global optimization algorithms. This manuscript, uses a CAMF (Cellular Automata Based Heuristic Solution Method for Minimizing Flow) algorithm to construct a ranking of candidate cells based on their associated forestation profit (sediment reduction) and cost of forestation (distance to roads). The two conflicting and concurrent objectives (maximize profit and minimize expense) allows us to model the situation as a Knapsack Problem. Variants of a Genetic Algorithm approach are proposed, using topology techniques as a heuristic, and the values obtained are compared with the value of the global optimum calculated with a Branch and Bound strategy. While Branch and Bound is shown to be superior to strategies such as Shani's PTAS algorithm integrated to CAMF-B. Branch and Bound is applicable to forestation problems with several thousands of cells. In situations with an excessive number of cells where it is not possible to apply Branch and Bound strategies due to computational costs, Genetic Algorithms always find a feasible solution with cost-benefit values close to the global optimums.

Key words: *CAMF, CAMF-B, Afforestation, Knapsack problem, Genetic algorithm, Branch and Bound, Topology*

Índice

Cláusula de Propiedad Intelectual

Yo, Luis Eduardo Lazo Vera, autor del trabajo de titulación "Resolución de un problema de selección de áreas óptimas para forestación, representado como un «Knapsack Problem», mediante la aplicación de un algoritmo genético y técnicas de Topología", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 26 de Octubre del 2018

Luis Eduardo Lazo Vera

C.I: 0301968186

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Yo, Luis Eduardo Lazo Vera en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Resolución de un problema de selección de áreas óptimas para forestación, representado como un «Knapsack Problem», mediante la aplicación de un algoritmo genético y técnicas de Topología",, de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 26 de Octubre del 2018

Luis Eduardo Lazo Vera

C.I: 0301968186

Epígrafe

Tu tiempo es limitado, de modo que no lo malgastes viviendo la vida de alguien distinto. No quedes atrapado en el dogma, que es vivir como otros piensan que deberías vivir. No dejes que los ruidos de las opiniones de los demás acallen tu propia voz interior. Y, lo que es más importante, ten el coraje para hacer lo que te dicen tu corazón y tu intuición.

—Steve Jobs

Agradecimiento

Consigno mi imperecedera gratitud a los distinguidos catedráticos: Ingeniero Lizandro Solano PhD, Subdecano de la Facultad de Ingeniería de la Universidad de Cuenca, por su motivación y apoyo brindados en la realización de esta obra, en calidad de Director, a quien considero mentor y guía en mi Carrera Académica; Doctor Wayne Materi PhD, eminente científico y ex-catedrático de Biología Molecular de la *University of Alberta* de Canadá; Ingeniero René Estrella PhD, por sus sugerencias que enriquecen esta tesis, y su valioso aporte al facilitar las bases de datos de la cuenca hidrográfica, tema de estudio de este proyecto; Ingeniero Pablo Vanegas PhD, Rector de la Universidad de Cuenca y autor del método *CAMF* y *CAMF-B*; en el que se fundamenta la primera parte de esta investigación; finalmente, al Máster Alejandro Izquierdo López, que cultiva con amor el arte y la ciencia, su interés por la Matemática, suscitó mi motivación científica, específicamente, por el mundo abstracto de la *Topología*

—Luis Eduardo

Dedicatoria

Con mucho amor y cariño, dedico esta esta obra, a mis queridos padres, Rosa y Segundo. Quienes marcaron mi sistema de valores, a los que nunca renunciaré.

A mi querida esposa Lorraine, su amor, ternura y compasión; me devolvió la vida y me hace sentir humano.

A mis queridos hermanos: Rosario, Ángel, José Antonio, Lourdes, Rómulo, Roberto, Segundo, Héctor y Juan Pablo, los llevo siempre en mi corazón.

—Luis Eduardo

Alcance y Objetivos

Alcance

Plantear un método eficiente para determinar las áreas de una cuenca hidrográfica que deberían ser forestadas, para minimizar la producción de sedimento, sin exceder el presupuesto predefinido.

Objetivos específicos

1. Representar el problema de selección de áreas a ser forestadas, para minimizar la producción de sedimento, mediante un «Knapsack problem» *(KP)*.
2. Proponer variantes del algoritmo genético Canónico, que resuelva el *KP* descrito en el numeral anterior de forma eficiente.
3. Comparación del rendimiento del algoritmo genético, descrito en el numeral anterior, con el rendimiento de un método exacto «Branch and Bound» para problemas pequeños.

Introducción

Desde una visión holística la naturaleza es un sistema armónico caótico, integrado e isomórfico al funcionamiento de un reloj, donde cada engranaje cumple una función especifica, susceptible y delicada a las perturbaciones del orden infinitesimal. El estudio de su comportamiento cobra especial trascendencia cuando tratamos de contribuir a su cuidado. Es por esta razón, iniciativas gubernamentales y asociaciones de protección del medio ambiente, en coordinación con universidades hacen esfuerzos para contribuir en su preservación.

La importancia que tiene el estudio de la producción de sedimento, desde un punto de vista ecológico, radica en el impacto que este fenómeno produce en los ecosistemas y la preservación del suelo, además de su impacto en ríos que alimentan represas que generan energía eléctrica o embalses para agua potable de las ciudades; la producción de sedimento es un fenómeno natural, que no se puede prever, pero puede ser atenuado en cierta medida, con múltiples estrategias como la forestación y manejo del suelo, para mejorar su resistencia a la erosión. Especialmente en las cuencas hidrográficas por su ubicación geográfica y la topografía del terreno, son áreas proclives a la pérdida del suelo, por la alta formación de sedimento (Molina et al., 2008). Una medida exitosa para el control del sedimento es la forestación, ayuda a mejorar la compactación del suelo y mejora la retención de agua, disminuyendo la producción de sedimento (FAO, 1979).

Las iniciativas de forestación en cuencas hidrográficas para la reducción de sedimento, buscan establecer las áreas óptimas, donde se debe plantar los árboles y administrar un presupuesto fijo; que debe ser invertido de tal forma que se obtengan los mayores beneficios del proyecto.

La información geográfica de las áreas que serán forestadas, se almacenan en mapas ráster, donde se pueden estudiar múltiples propiedades, cada celda en el mapa representa una área potencial en donde se plantara bosque y se obtendrán beneficios (reducción de sedimento), naturalmente forestar esta área en la celda debe tener asociado un costo de forestación, que se debe restar del presupuesto establecido. Por esta razón existen

métodos como: el método de solución heurística basada en autómatas celulares para minimizar el flujo (del inglés Cellular Automata Based Heuristic Solution Method for Minimizing Flow, CAMF) y el método de solución heurística basado en autómatas celulares para minimizar el flujo sujeto a restricción presupuestario (del inglés CA-Based Heuristic Method for Minimizing Flow Under a Budget Constraint CAMF-B), introducido por (Vanegas et al., 2010) y On-site CAMF, introducido por (Estrella, 2015), etc. Estos métodos tratan de brindar herramientas de soporte en la selección de las áreas que deben ser forestadas con el fin de obtener los mejores beneficios (reducción de sedimento). No obstante se ha demostrado que la complejidad computacional de *CAMF* es proporcional al tamaño de las bases de datos (Estrella, 2015), lo cuál es una desventaja para *CAMF*. El algoritmo *CAMF-B*, integra estrategias del «Knapsack Problem» y *Shani's PTAS algoritmo* de complejidad $O(n^{k+1})$, lo cuál demanda gran cantidad de recursos de memoria, limitando su aplicabilidad a problemas de forestación con gran cantidad de áreas (Vanegas et al., 2010).
En este proyecto buscamos integrar nuevas estrategias que mejoren la selección de áreas óptimas, combinando *CAMF, «Knapsack Problem», Branch and Bound, Algoritmo Genético y deformación topológica*.

Organización del manuscrito

El Capítulo 1 presenta la motivación y literatura relevante sobre los problemas de la erosión del suelo, estrategias de control y métodos que ayudan en la selección de áreas óptimas, donde se forestará, también incluye una descripción de la problemática y estrategias que se integrarán.
En el Capítulo 2 se presenta literatura sobre los problemas de tipo *NP-fuertes* y *NP-completos*, «Knapsack Problem», *CAMF, CAMF-B, Branch and Bound, Algoritmo Genético, Topología y deformaciones*. Tratan de introducir los conceptos que se aplicarán en los capítulos posteriores y analizar el estado actual del conocimiento científico de dichas estrategias.
El Capítulo 3 describe la región de estudio (cuenca hidrográfica), materiales y métodos utilizados, modelación de los problemas de forestación cómo un «Knapsack Problem», y la implementación de los algoritmos *Genético* y *Branch and Bound*.
En el Capítulo 4 se presenta los resultados obtenidos y una discusión de los mismos.
En el Capítulo 5 se presenta las conclusiones.

Capítulo 1

Motivación y descripción del problema

1.1. Sedimento y erosión del suelo

La producción de sedimento en cuencas hidrográficas conlleva múltiples problemas como: deslizamientos del terreno, pérdida paulatina de la fertilidad agrícola y alteración del ecosistema de los ríos, al bloquear la luz solar, etc (Tetsuya Kusuda, 2007). El sedimento, al desembocar en ríos que alimentan represas o estanques para riego, se deposita en el fondo de los reservorios; alterando la capacidad verdadera de los estanques. La erosión del suelo causada por la producción de sedimento es muy crítica en cuencas hidrográficas, por su ubicación geográfica y la topografía del terreno, se sabe que en terrenos con pendientes pronunciadas el flujo del agua siempre va a seguir hacia cotas inferiores, llevando consigo partes del suelo que no se pueden recuperar. La pérdida del suelo se puede deber a múltiples factores que causan el desprendimiento de las partículas del suelo, como: la lluvia excesiva causa el deslizamientos del suelo, el viento también contribuye a la erosión al quitar las partículas de suelo, cambios químicos en la estructura del suelo como aumento de salinidad, puede esterilizar el suelo, factores biológicos como el incremento de humus causada por microorganismos, etc (FAO, 1979).

La erosión del suelo es un problema que se trata de atenuar desde la creación misma de la agricultura, civilizaciones enteras en la historia de la humanidad han desarrollado diferentes técnicas para el control de la erosión del suelo, la cultura Cañari y Azteca enfrentaban este fenómeno con los sistemas de terrazas para evitar la pérdida del suelo, en zonas montañosas con pendientes pronunciadas (Sánchez y Pérez, 2013), donde plantaban sus cultivos.

1.1.1. Métodos de control

La erosión es un proceso natural que no se puede prever, y su control es un proceso complejo en donde convergen muchos parámetros a considerar, como: ubicación geográfica, tipo de terreno, uso del suelo, topografía, etc. Acorde a esta estructura formal, se podría considerar estrategias, como: i) la forestación que ayuda a mejorar la retención del agua y disminuye la producción de sedimento (Palmieri et al., 2001); ii) la manipulación directa del terreno alterando su topografía y administración del suelo de tal forma que sea más resistente a la erosión, etc (FAO, 1979). La estrategia más usada cuando se trata de prevenir la erosión del suelo en cuencas hidrográficas, destaca sobre todo la forestación (Yoo et al., 2014), por esta razón iniciativas gubernamentales y asociaciones de protección del medio ambiente impulsan los procesos de forestación con el fin de evitar la pérdida del suelo, contribuir a preservar los ecosistemas y preservar el agua.

1.1.2. Representación de la áreas a ser forestadas

Cuando las áreas a ser forestadas ocupan grandes extensiones de terreno como es en el caso de cuencas hidrográficas de varios kilómetros, esta información se maneja en mapas ráster (Estrella, 2015), que son imágenes digitalizadas. Estas imágenes están formadas por una matriz rectangular, donde cada celda contiene información sobre sí misma y está relacionada a cierta área del mapa. Una capa ráster está formada por cuatro componentes fundamentales: matriz de datos, información geométrica de la matriz, tabla de colores y etiquetas para información cualitativa en caso de existir (Fiume, 2014). Esto facilita el manejo de la información y permite estudiar múltiples propiedades de las áreas donde se pretende plantar los bosques como: ubicación, dirección y flujo del sedimento, producción de sedimento, indice de permeabilidad, saturación, altitud, pendiente, etc.

1.1.3. Métodos aplicados en proyectos de forestación y problemática

Un aspecto fundamental en todo proyecto de forestación, es la existencia de un presupuesto limitado que debe ser invertido de manera que se obtenga los mayores beneficios posibles, cuando se trata de problemas de forestación con el fin de reducir la producción de sedimento en cuencas hidrográficas, normalmente la pregunta es:

¿Cuáles son las áreas dentro de la cuenca que deberían ser forestadas?, determinar de una manera científica las áreas candidatas a ser forestadas, por otro lado **¿Cómo sabemos si las áreas forestadas, resultan, ser las óptimas?**, de tal forma que se obtenga el mejor beneficio (reducción de sedimento), ciertas áreas contribuyen con mayor retención de sedimento que otras (Vanegas et al., 2010). Por este motivo existen métodos como: el método de solución heurística basada en autómatas celulares para minimizar el flujo (del inglés Cellular Automata Based Heuristic Solution Method for Minimizing Flow, CAMF) y el método de solución heurística basado en autómatas celulares para minimizar el flujo sujeto a restricción presupuestario (del inglés CA-Based Heuristic Method for Minimizing Flow Under a Budget Constraint CAMF-B), introducido por (Vanegas et al., 2010) y On-site CAMF, introducido por (Estrella, 2015), etc. Que tratan de responder las dos interrogantes, y establecer de manera científica dónde están las mejores áreas.

CAMF-B integra estrategias del «Knapsack Problem», para seleccionar las celdas de la cuenca hidrográfica que deberían ser forestadas, de manera eficiente, no obstante el «Knapsack Problem» es un problema del tipo *NP-Hard*, y la heurística que integra *CAMF-B* es de complejidad computacional de orden $O(n^{k+1})$ lo cuál restringe la aplicabilidad de este método a problemas de forestación donde interviene un número excesivo de áreas a seleccionar (Vanegas et al., 2010). Dentro de esta estructura formal, esta investigación busca probar nuevas estrategias, que complementen el conocimiento presente, integrando estrategias que mejoren la selección de áreas óptimas, combinando *CAMF, «Knapsack Problem», «Branch and Bound», Algoritmo Genético y deformación topológica*.

Capítulo 2

Revisión de literatura

2.1. Problemas del tipo NP-fuertes y NP-completos

Los problemas del tipo NP-fuertes y NP-completos están relacionados directamente con la complejidad algorítmica requerida para su solución. Existen problemas que pueden ser resueltos utilizando un algoritmo que se ejecuta en tiempo polinomial y problemas en los cuales no se conoce un algoritmo de tiempo polinomial que pueda resolverlo. Los mejores algoritmos que se han estudiado, se podrían clasificar en dos grupos: 1) problemas cuyo tiempo de ejecución está acotado por un polinomio de grado pequeño como por ejemplo: búsqueda ordenada en cuyo caso su tiempo de ejecución es de $O(\log n)$, evaluación polinomial en cuyo caso es de $O(n)$, *Quicksort* $O(n \log n)$ y *string matching* cuyo caso es $O(mn)$ y 2) problemas cuyo mejor algoritmo conocido es polinomial; ejemplos de estos problemas son: problema del viajante y «Knapsack Problem» para los cuales los mejores algoritmos tienen complejidad $O(n^2 2^n)$ y $O(2^{n/2})$, respectivamente. En lo que respecta a los problemas del segundo grupo, en la actualidad no se ha logrado desarrollar algoritmos eficientes de tiempo polinomial que puedan emplearse para su resolución (Rashid, 2010).
Todo problema del tipo *NP-completos* son del tipo *NP-fuertes* no obstante solo se conoce unos pocos problemas del tipo *NP-fuertes* que son del tipo *NP-completos*. Si un problema del tipo *NP-fuerte* se puede resolver en tiempo polinomial todos los del tipo *NP-completos* pueden resolverse en tiempo polinomial (Karp, 1972).

2.2. «Knapsack Problem»

El problema de la mochila (del inglés «Knapsack Problem» KP) es un problema de optimización combinatoria, en el cual se tiene una mochila (knapsack) con una

capacidad y objetos asociados a un beneficio y a una capacidad. Se requiere colocar en la mochila un combinación de objetos que maximice el beneficio; sin exceder la capacidad de la mochila. Si numeramos los objetos de $1,...n$ e introducimos un vector de variables enteras $x_j(j = 1,...n)$ de tal forma:

$$x_j = \begin{cases} 1 & \text{Si el objeto j es seleccionado como parte de la solución.} \\ 0 & \text{Caso contrario.} \end{cases} \tag{2.1}$$

y si definimos como p_j la medida del beneficio de seleccionar el objeto j con capacidad asociada w_j; teniendo la capacidad de la mochila C. Entonces la solución del problema será un subconjunto disjunto escogido de todas las posibles combinaciones del vector x_j sujeto a la restricción:

$$\sum_{j=1}^{n} w_j x_j \leq C \tag{2.2}$$

y maximice el beneficio

$$\sum_{j=1}^{n} p_j x_j \tag{2.3}$$

Desde un punto de vista práctico, la importancia del **KP** se ve reflejado en la facilidad de integrar su filosofía a problemas de optimización, en áreas como: control de presupuesto, logística, transporte, control de stock, etc (Martello, 1990). El **KP** es un problema del tipo *NP-completo* (Karp, 1972) y su mejor algoritmo que se ejecuta en tiempo polinomial es de complejidad $O(2^{n/2})$ (Rashid, 2010).
Para resolver el KP es posible emplear métodos exactos como: enumeración exhaustiva, programación dinámica, «Backtracking» y Ramificación y Acotación (del inglés, Branch and Bound) (Rashid, 2010). La desventaja de estos métodos radica en que sus requerimientos computacionales se vuelven prohibitivos para todo **KP** de un tamaño mediano o grande, como se evidencia en la Tabla 2.1 para el caso de enumeración exhaustiva, se deberá verificar cada una de las posibles soluciones para garantizar la obtención del óptimo global, por lo cual el algoritmo se vuelve inaplicable dado el tamaño del conjunto solución.

Tiempo para evaluar $f(n)$ instrucciones. En un computador de 10^9 instrucciones/segundo	
n	$f(n) = 2^n$
10	$1\mu s$
20	$1ms$
30	$1s$
40	$18{,}3m$
50	$13d$
100	$4E13$ años
1000	$32E283$ años

Tabla 2.1: Donde n representa el número de objetos y $f(n)$ define el número de posibles soluciones (Rashid, 2010).

Dado que el **KP** es un problema de optimización del tipo **NP-Hard** se han implementado diferentes algoritmos aproximados: ε-*approximation algorithm*, Shani's **PTAS** algoritmo cuya complejidad computacional es de orden $O(n^{m+1})$ lo que demanda gran cantidad de memoria para «Knapsacks» de tamaños (número de objetos) medianamente grandes (Martello, 1990).

2.2.1. 0-1 unidimensional «Knapsack Problem»

El 0-1, o binario «Knapsack Problem» (KP) requiere un conjunto de objetos que deben ser colocados en una mochila («Knapsack») de una capacidad dada (C), cada objeto (j) está asociado a un beneficio (p_j), y un peso (w_j). Se requiere maximizar la suma de los beneficios de los objetos colocados en la mochila de tal forma que la suma de los pesos asociados a los objetos no exceda la capacidad de la mochila (Martello, 1990). Formalmente definido por la ecuaciones (2.4):

$$\begin{aligned} &Maximizar \quad \sum_{j=1}^{n} p_j x_j \\ &Sujeto\ a: \\ &\sum_{j=1}^{n} w_j x_j \leq c \quad x_j \in \{0,1\} \forall j \quad , j \in N \end{aligned} \tag{2.4}$$

2.2.2. Multidimensional «Knapsack Problem»

El Multidimensional «Knapsack Problem» (MKP) es una generalización del 0-1 unidimensional «Knapsack Problem». El problema consiste en múltiples mochilas («Knapsacks»). Requiere un conjunto de objetos que deben ser colocados en las mochilas («Knapsacks») asociada a una capacidad (C_i), cada objeto $(j = 1,...n)$ está asociado a un beneficio (p_j), y un peso (w_{ji}) con $(i = 1,...m)$. Se requiere maximizar la suma de los beneficios de los objetos colocados en las mochilas de tal forma que la suma de los pesos asociados a los objetos no excedan las capacidad de las mochilas (Martello, 1990). Formalmente definido por las ecuaciones (2.5):

$$\begin{aligned} &Maximizar \quad \sum_{j=1}^{n} p_j x_j \\ &Sujeto\ a: \\ &\sum_{j=1}^{n} w_{ij} x_j \leq C_i \quad x_j \in \{0,1\} \forall j \quad i,j \in \mathrm{N} \end{aligned} \tag{2.5}$$

Para solucionar este tipo de «Knapsacks» se pueden emplear los mismos algoritmos para el 0-1 unidimensional «Knapsack Problem», no obstante la complejidad computacional se ve comprometida por lo que se suele optar por heurísticas o sistemas híbridos, como la combinación del Algoritmo Genético y *Branch and Bound* (Michalewicz y Hartley, 1996).

2.3. CAMF

El método de solución heurística basada en autómatas celulares para minimizar el flujo (del inglés, Cellular Automata Based Heuristic Solution Method for Minimizing Flow, CAMF) es un método introducido por (Vanegas et al., 2010) el cual busca determinar de una manera científica las mejores áreas dentro de una cuenca hidrográfica que se deberían forestar para minimizar la producción de sedimento. El método **CAMF** está basado en autómatas celulares e integra interacción espacial: cuando una celda es forestada tiene un impacto directo en las celdas vecinas, además las propiedades de esta celda y sus vecinas cambia, alterando la producción de sedimento, retención y salida de sedimento en la celdas de la vecindad.

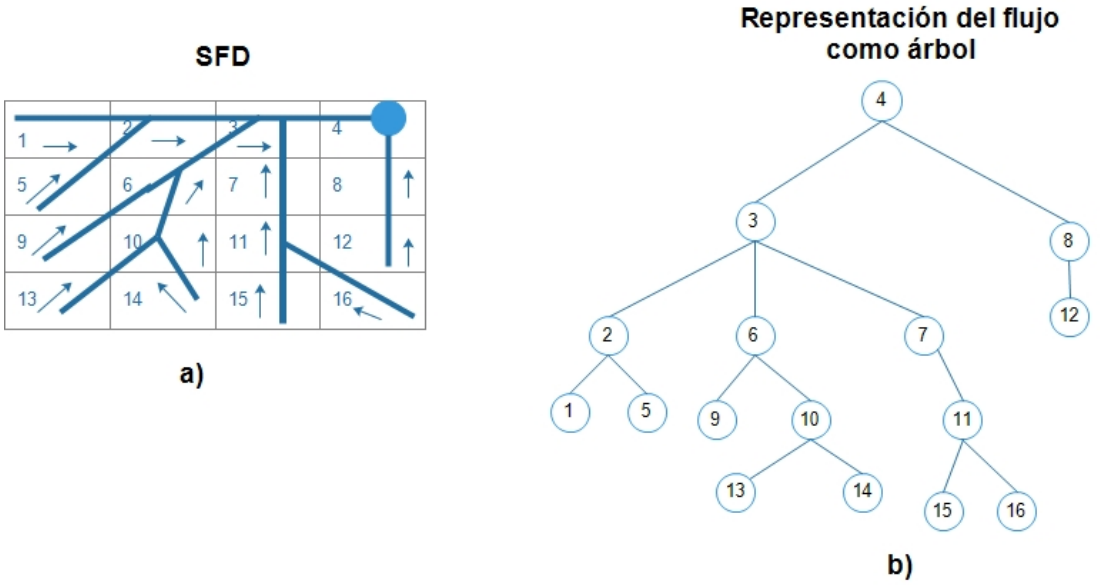


Figura 2.1: Dirección de flujo unidireccional, a) representación matricial, b) representación en árbol

2.3.1. Modelo de flujo unidireccional

El modelo de flujo unidireccional (del inglés, Single Flow Direcction SFD) está basado en los métodos desarrollados por (Jenson y Domingue, 1988), este modelo compara la elevación de cada celda en un mapa ráster con su vecindad y traza una ruta continua única hacia la celda vecina, luego ésta se conecta de forma continua hacia sus vecinas hasta converger en una única salida como se puede ver en la Figura 2.1, en a) las flechas indican flujo hacia la salida y las líneas continuas indican la representación del flujo del árbol b). Se puede observar, como todos los flujos de las celdas que se encuentran ubicadas en las cotas superiores, convergen a una única salida (Wolock y McCabe Jr, 1995).

2.3.2. Autómata Celular

Autómata Celular (A.C) representa uno de los modelos más importantes para simulación de procesos dinámicos complejos donde existe interacción espacial, introducida por (Von Neumann et al., 1966). Algunos autores definen A.C como un modelo geográfico (Tobler, 1979) que opera en una matriz de celdas, donde la información almacenada en las celdas, cambia con el tiempo y esta dado en función de su estado inicial y el estado de su vecindad. En cada celda (i) se almacena información relevante del fenómeno físico en cuestión en un instante de tiempo t, el estado de la celda en el instante de tiempo $(t+1)$ está en función del su estado inicial en el tiempo t, y del estado de su

vecindad. Se define el estado de la celda i en el tiempo t como (S_i^t) y el estado de la vecindad como (Ω_i^t), entonces, el instante de una celda i en el tiempo $t+1$ se define por la ecuación (2.6)

$$S_i^{(t+1)} = f(S_i^t, \Omega_i^t) \tag{2.6}$$

2.3.3. Función definida por partes

El algoritmo **CAMF** utiliza una función definida por partes para modelar la cantidad de sedimento que una celda desplaza a las celdas vecinas. Esta función representa la relación de la acumulación efectiva (del inglés, *Effective Accumulation EA*) como se muestran en la Figura 2.2 donde se puede ver los estados de una celda antes y después de la forestación definida por los tres segmentos de esta función: el primer segmento correspondiente a los valores que se encuentran desde $EA = 0$ hasta la capacidad de retención de la celda, que en la figura corresponde al primer punto denotado como σ_1, por lo tanto si la producción de sedimento en la celda (j) está por debajo de su capacidad de retención; se concluye que la celda no desplaza sedimento. El segundo segmento se encuentra desde la capacidad de retención σ_1 hasta el punto de saturación σ_2 si la celda presenta un valor hasta el punto de saturación, se concluye que la celda expulsará parte del sedimento que es directamente proporcional al factor de flujo, en el caso de la celda sin forestación este factor de flujo corresponde a la pendiente de la celda y finalmente el tercer segmento cuando el sedimento sobrepasa el punto de saturación se concluye que todo el sedimento será desplazado a la celda vecina.
Los parámetros, con los que la función definida por partes trabaja, se espera que cambien con la forestación. De esta forma, los umbrales de: saturación, retención y factor de flujo serán más altos y la producción de sedimento será mucho menor, como se muestran los resultados en la Figura 2.2, para antes y después de la forestación; b) la capacidad de retención(σ_3) es mayor que en a).

2.3.4. Datos de entrada para CAMF

El método **CAMF** requiere un conjunto de mapas ráster con diferentes características de la cuenca hidrográfica, antes de que las celdas sean forestadas, de esta manera puede comparar los dos estados, antes y después, de la forestación. Además de los mapas ráster **CAMF** necesita un número fijo de celdas que se requieren ser forestadas, la salida del algoritmo, es una lista de celdas ordenadas, las cuales deben ser forestadas

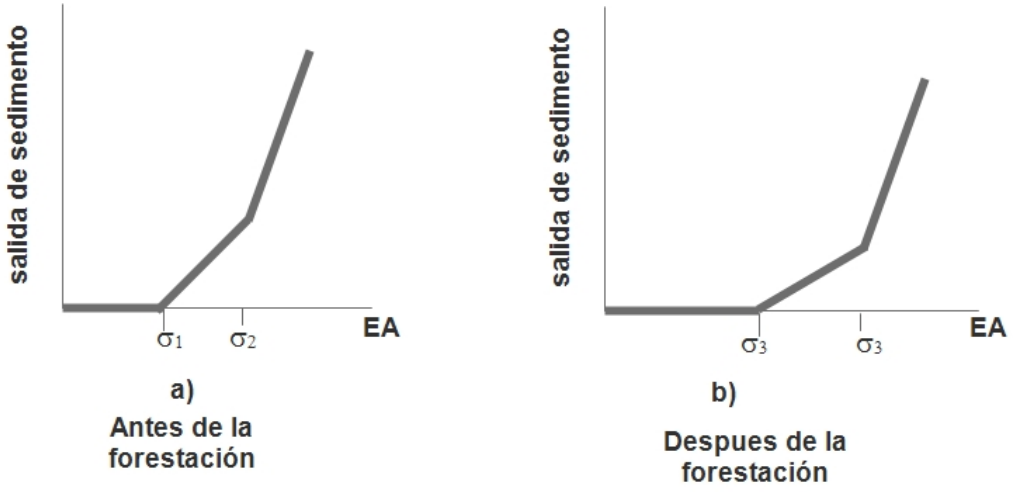


Figura 2.2: a) Estado de una celda que no ha sido forestada; b) Estado de la celda después de la forestación (Vanegas et al., 2010)

con el fin de minimizar la acumulación de sedimento en la salida de la cuenca.
El método **CAMF** para su ejecución requiere:

1. Un mapa ráster con la producción de sedimento: cada celda produce sedimento expresada en $kg/km^2\ yr^{-1}$

2. Capacidad de retención: cada celda tiene asociada una capacidad de retención expresada en $kg/km^2\ yr^{-1}$, si la producción de sedimento de la celda es menor que la capacidad retención entonces esta celda no desplaza sedimento a ninguna celda vecina.

3. Umbral de saturación: toda celda cuya producción de sedimento sobrepase el nivel de saturación tiene que desplazar sedimento expresado en $kg/km^2\ yr^{-1}$ a la celda con más pendiente de inclinación hacia abajo.

4. Factor de flujo: este *dataset* indica la fracción de sedimento que desplaza una celda hacia su vecina. Este factor se aplica únicamente cuando la producción de sedimento se encuentra en el intervalo entre la capacidad de retención y el umbral de saturación

5. Mapa ráster con dirección de flujo en una sola dirección (del inglés Single Flow Direction raster map) de la cuenca en cuestión, indica el camino que tiene que seguir el sedimento hasta la salida de la cuenca.

6. Número de celdas: este parámetro indica el número de celdas que deben contener la solución, el método **CAMF** construirá una lista de este tamaño con las mejores celdas candidatas a ser forestadas.

CAMF requiere de dos versiones de los *datasets* indicados: el primero corresponde a los datos de la cuenca sin forestación y el segundo corresponden a la cuenca con forestación. Estas dos versiones de los datos son utilizados por **CAMF** para calcular la cantidad de sedimento que desplaza cada celda.

2.3.5. Funcionamiento del algoritmo CAMF

El algoritmo **CAMF** realiza los siguientes pasos:

1. Calcula la acumulación de sedimento de cada celda, como la suma del sedimento producido por la propia celda y el sedimento que llega de las celdas vecinas.

2. Para cada celda candidata se calcula la reducción de sedimento en el caso de ser forestada.

3. Construye una lista con las celdas candidatas y el beneficio que representa de ser forestada (reducción de sedimento) de la lista construida en el paso 2.

4. Únicamente las celdas que se encuentran en el ranking más alto son seleccionadas como parte de la solución, al contribuir con los mejores beneficios.

5. La acumulación de sedimento debe ser actualizada para la celda forestada y para todas las las celdas vecinas, así como para las celdas que se encuentran en a la salida de la cuenca.

6. Si el número de celdas que integran la lista de celdas candidatas es menor que el número de celdas requeridas, se repite el paso 2 hasta alcanzar el valor requerido.

2.3.6. CAMF-B

El método de solución heurística basado en autómatas celulares para minimizar el flujo sujeto a restricción presupuestario (del inglés CA-Based Heuristic Method for Minimizing Flow Under a Budget Constraint CAMF-B) es una variante de **CAMF** (Vanegas et al., 2010) para restringir la selección de celdas que serán forestadas, en función de un presupuesto fijo, es una variante de **CAMF**, que considera la restricción del presupuesto para seleccionar las celdas a ser forestadas, en lugar de emplear un

número fijo de celdas; e integra estrategias del «**Knapsack Problem**». Combinando estas dos estrategias, cada celda está asociada a un costo de forestación, que se mide, en unidad de celda y un beneficio (reducción de sedimento en la salida de la cuenca) que está dada por la aplicación de CAMF. Para resolver el «**Knapsack**» resultante, **CAMF-B** integra *Sahni's PTAS algorithm* (Sahni, 1975).

2.3.7. Funcionamiento de algoritmo CAMF-B

Para ejecutar **CAMF-B** a parte de los datos de entrada necesarios para ejecutar **CAMF**, se requiere un mapa ráster con los costos de forestación asociadas al área de estudio. La diferencia fundamental de **CAMF-B** radica en la restricción del presupuesto asociado a las celdas que serán forestadas, en **CAMF** únicamente se requería de un número fijo de celdas a ser forestadas.
CAMF-B consta de los siguientes pasos:

1. Calcula la producción de sedimento *(EA)* de cada nodo en la representación de árbol (Figura 2.1 *b*)
2. Calcula la diferencia óptima del flujo a la salida de la cuenca.
3. Selecciona los nodos a ser forestados aplicando *Sahni's PTAS algorithm* para el «Knapsack Problem».

El problema de este método radica en la complejidad computacional del algoritmo Shani's **PTAS algoritmo** que se integra a CAMF-B es de $O(n^{k+1})$ (Martello, 1990) lo cual demanda gran cantidad de memoria para problemas de forestación donde el número de celdas es muy grande debido a la resolución que se utiliza.

2.4. Algoritmo Genético

El Algoritmo Genético Canónico fue introducido por Jhon Holland de la Universidad de Michigan e integra dos componentes fundamentales, la naturaleza genética y la selección natural. El algoritmo genético parte de una población inicial, luego esta población pasa por un proceso de selección, donde los mejores genes tienen mayor probabilidades de transferir su información genética a las siguientes generaciones; la combinación de estos genes se conoce como cruzamiento y tiene el objetivo de mezclar información genética de los mejores individuos de la población para generar descendencia (offspring). La mutación de algún gen se aplica con una probabilidad baja, permitiendo diversificar la exploración del espacio solución; también se suele introducir

a la nueva generación los mejores genes de los padres, esto se conoce como elitismo y preservar buena información genética de las generaciones anteriores (DeJong, 1975).

2.4.1. Áreas de aplicación del Algoritmo Genético

El Algoritmo Genético al ser una metaheurística tiene diferentes áreas de aplicación, entre las cuales destacan:

1. Diseño y aprendizaje de redes neuronales.
2. Diseño de itinerarios.
3. Teoría de juegos.
4. Robótica.
5. Diseño de estructuras.
6. Análisis de señales.

2.4.2. Fundamento biológico

Todo ser vivo está compuesto de células; cada célula esta compuesta de cromosomas que son cadenas de *Ácido desoxirribonucleico (ADN)* que representa el modelo de todo el organismo. Un cromosoma esta constituido por bloques de genes; cada uno de estos genes codifica un rasgo único por ejemplo; color de ojos, altura, predisposición a ciertas enfermedades, etc. Un valor especifico de este gen como por ejemplo (ojos color negro), se conoce como *alleles*. Cada gen ocupa un lugar especifico en el cromosoma llamada *locus*. Una secuencia completa de material genético se conoce como *genome* y una partición de genes en el *genome* se conoce como *genotype*. El *genotype* depende del *phenotype*, características físicas y mentales desarrolladas antes de nacer como, inteligencia, color de ojos, etc.

2.4.3. Creación de descendencia

Durante la creación de descendencia (del inglés *offspring*) ocurre una recombinación del material genético (del inglés *cross over*) proceso durante el cual se combina la información genética de los padres creando un cromosoma nuevo en cual puede sufrir mutaciones. La mutación (del inglés *mutation*) se da cuando la cadena de *ADN* sufrió alguna alteración en la secuencia de los genes que conforman el cromosoma. El *fitness* de un organismo se refiere a una medida de supervivencia del organismo en vida.

2.4.4. Operadores fundamentales del Algoritmo Genético

El Algoritmo Genético integra tres operadores básicos:

- Reproducción.
- Cruzamiento (del inglés *Crossover*).
- Mutación.

Reproducción

Es el primer operador que se aplica a la población inicial con el fin de seleccionar a los candidatos más aptos, este operador es llamado a veces *operador de selección*, se fundamenta en la teoría de la evolución de las especies de Charles Darwin, los mejores individuos son seleccionados para generar descendencia (*offspring*), en la literatura científica existe muchos operadores de selección no obstante entre los más usados destacan:

1. Selección por ruleta.
2. Selección por torneo.

Selección por ruleta

Este operador de selección se basa en la probabilidad de supervivencia de cada individuo, los más aptos tienen mayor probabilidad de contribuir con más copias de su material genético, el algoritmo se describe en la Tabla 2.2, donde n =tamaño de la población, una vez calculado estos valores, tenemos que generar un número aleatorio k que se encuentre en el intervalo cerrado [0,1] si $k < q_1$ entonces seleccionar el primer cromosoma, caso contrario seleccionar el i-ésimo cromosoma $v_i(2 \leq i \leq m)$ sujeto a $q_{i-1} < k \leq q_i$.

Selección por ruleta	
//Calcular el valor del fitness	
$evaluar(v_i)$	$\forall v_i (i = 1, ...m)$
//Sumar el fitness de toda la población	
$F = \sum_{i=1}^{n} evaluar(v_i)$	$\forall v_i (i = 1, ...m)$
//Calcular la probabilidad de selección p_i de cada cromosoma	
$p_i = evaluar(v_i)/F$	$\forall v_i (i = 1, ...m)$
//Calcular la probabilidad acumulada q_i para cada cromosoma	
$q_i = \sum_{i=1}^{i} p_j$	$\forall v_i (i = 1, ...m)$

Tabla 2.2: Algoritmo selección por ruleta (Rashid, 2010).

Selección por torneo

El operador de selección únicamente elige a los mejores individuos para generar descendencia (*offspring*), el objetivo fundamental del operador de selección es favorecer la conservación de los mejores genes y que este material genético mejore en las futuras generaciones. En todo proceso de búsqueda que implica la aplicación de procesos evolutivos se debe satisfacer dos objetivos fundamentales: diversidad de la población y selección de los mejores individuos (Luke, 2013). Esto implica que de los nuevos individuos, se espera que aporten a mejores áreas de exploración, y la selección, el grado de que tan buenos son estos individuos.
Entonces mientras más estricto es la selección, la convergencia es mucho más rápida no obstante se esto puede llevar a convergencia prematura en un óptimo local, debido a la pérdida de la diversidad del espacio de exploración. Si la selección es demasiado blanda, la convergencia a una buena solución es muy lento debido a la diversidad del espacio de búsqueda (Luke, 2013).

Por lo tanto un buen operador de selección debe permitir un balance entre estos dos objetivos concurrentes *diversidad* y *selección*. Dentro de este contexto el operador de selección por ruleta presenta dos problemas:

- Estancamiento de la búsqueda, al integrar una selección demasiada fuerte e inclusive como está basado en probabilidades algunos individuos resultarán duplicados.
- Convergencia prematura debido a la reducción del espacio de búsqueda.

La selección por torneo, por otro lado selecciona a los individuos de manera aleatoria,

se selecciona de la población los individuos que van a competir, entonces se calcula el valor de *fitness* luego el que tenga el valor más alto es el ganador y es seleccionado para formar parte del grupo que generará después *offspring*.
Esta forma de seleccionar brinda mayor diversidad en la población, y permite tener buenos individuos como padres de la nueva generación (Luke, 2013).

Crossover

Una vez concluido el proceso de selección en la fase de reproducción se realiza una clonación de los individuos seleccionados en lugar de generar nuevos.
El operador *crossover* es aplicado a la población con el objetivo de crear un nuevo individuo. El objetivo fundamental de este operador es explorar de forma paramétrica el espacio de búsqueda. Al combinar los genes de dos individuos buenos se espera crear un individuo con mejores características que los padres.
El procedimiento para elegir los individuos cuyo material genético serán combinados requiere de los siguientes pasos.

- Establecemos una probabilidad de cruzamiento p_c de valor constante h, $p_c \leftarrow h$.
- Para cada cromosoma ganador asignamos un número aleatorio k, $k \leftarrow [0,1]$ si $k < p_c$ entonces se selecciona el cromosoma para el cruzamiento.

Una vez realizado la selección de las parejas se debe establecer un punto de corte (del inglés *pos*) para cada pareja, *pos* es un número entero aleatorio en función de la longitud del cromosoma (número de bits usados para representar $[1,...n]$) donde $n = longitud\ del\ cromosoma$.
Entonces una vez establecido *pos* se tiene los dos cromosomas:

$$(a_1, a_2, ...a_{pos}, a_{pos+1}...a_n)$$
$$(b_1, b_2, ...b_{pos}, b_{pos+1}...b_n)$$

Después del cruzamiento los cromosomas resultantes son:

$$(a_1, a_2, ...b_{pos}, b_{pos+1}...b_n)$$
$$(b_1, b_2, ...a_{pos}, a_{pos+1}...a_n)$$

Con esta misma filosofía la literatura científica presenta varios tipos de cruzamiento que se resumen en la figura 2.3:

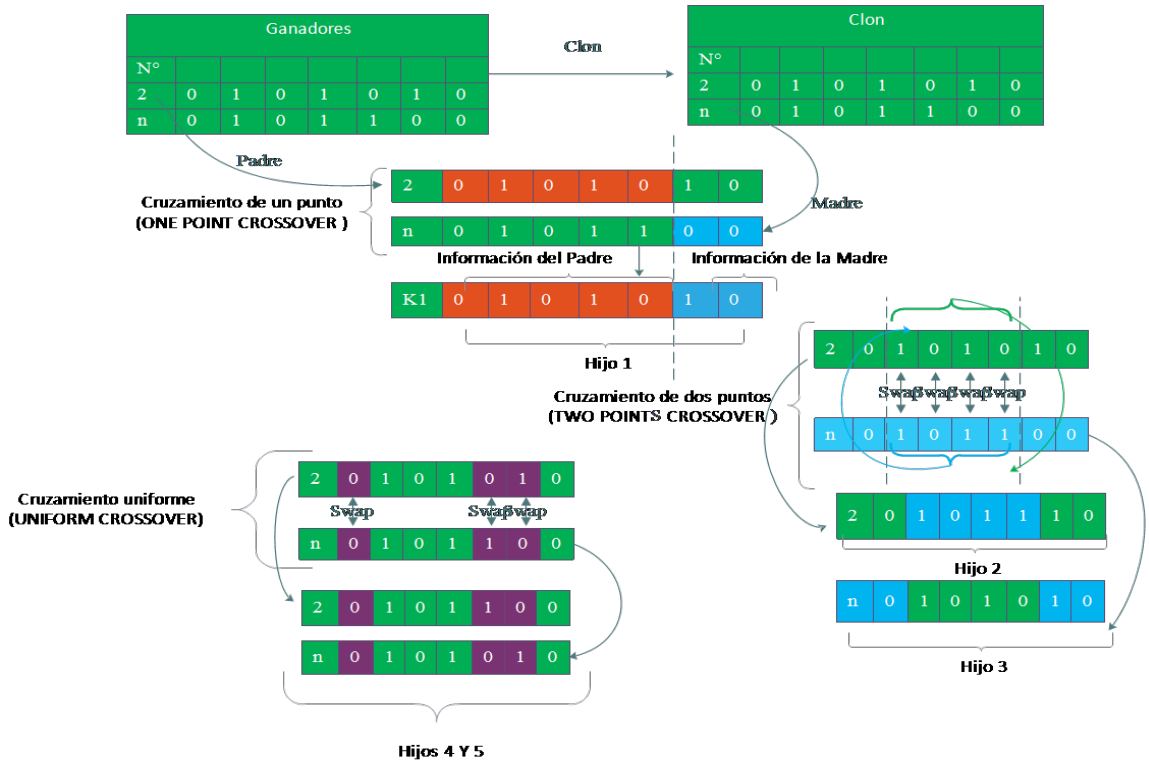


Figura 2.3: Principales operadores de cruzamiento del Algoritmo Genético

Mutación

Este operador está en función de una probabilidad constante que recibe como parámetro el Algoritmo Genético. Y se aplica después de la reproducción, su objetivo es diversificar el espacio de búsqueda, ningún otro operador del Algoritmo Genético puede hacer cambios directos a los genes del cromosoma excepto el de mutación. Entonces el operador de mutación tiene el principal objetivo en cierta medida atenuar este fenómeno, evitando la convergencia en mínimos locales y diversificar el espacio de búsqueda. En el Algoritmo Genético se debe configurar a priori la probabilidad de mutación p_m que especifica el porcentaje de genes que van a ser mutados dentro del cromosoma, se recomienda probabilidades inferiores al 1 %. Después de aplicar el operador *crossover* cada gen del cromosoma tiene las mismas probabilidad de ser mutado. Para cada gen en el cromosoma se genera un número aleatorio k en el intervalo cerrado $[0, 1]$ si $k < p_m$ el gen se muta, en la Figura 2.4 se expone la forma de aplicar este operador.

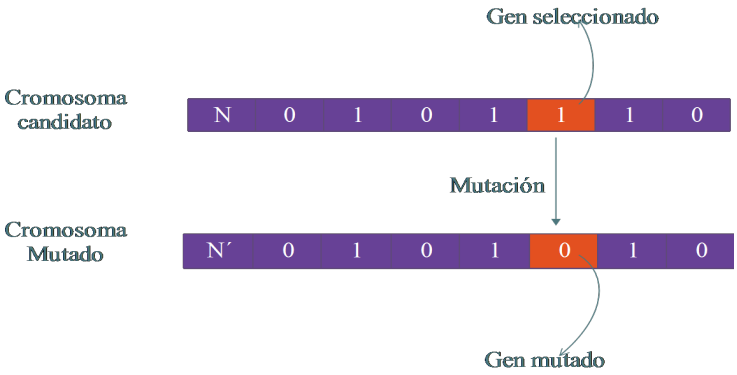


Figura 2.4: Mutación

2.4.5. Parámetros para terminar la ejecución del algoritmo

El Algoritmo Genético al ser una metaheurística y considerando que estamos frente a problemas de optimización, es importante determinar el tiempo que se va ha invertir en la ejecución del algoritmo; normalmente una heurística es como un explorador que camina dentro del espacio de búsqueda, se encontrará con buenas soluciones, malas, o hasta con el óptimo global, no obstante resulta imposible asegurar la obtención del valor óptimo. Dentro de esta estructura formal, la literatura científica (Michalewicz y Hartley, 1996) (Luke, 2013) concerniente al Algoritmo Genético recomiendo tres parámetros que pueden ser configurados para detener la búsqueda:

- Número de generaciones.
- Convergencia de la población.
- Convergencia de los genes.

Particularmente algunos autores recomiendan optar por el número de generaciones (Rashid, 2010).

2.5. Topología

La topología como ciencia abstracta permite clasificar estructuras matemáticas complejas como familias. Dos objetos matemáticos que permiten representación topológica de la una sobre la otra mediante transformaciones continuas, se las llama homeomorfas (Threlfall., 1951). El homeomorfismo desempeña la parte más importante en el área de la topología, cuando dos objetos son transformables unos en otros mediante torsión y distensión se dice que son homeomorfas (se establece el homeomorfismo) (Tokieda,

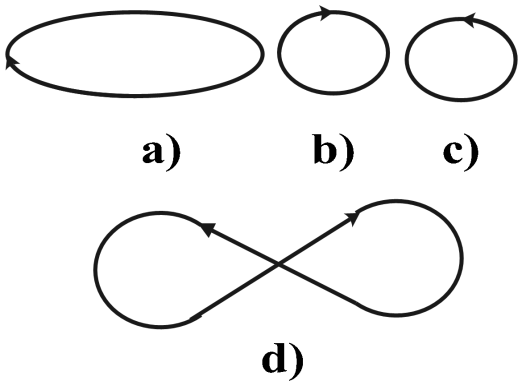


Figura 2.5: Las curvas a) y b) son isotópicas, las cuatro son homotopas y a),b),c) son homeomorfas

2001). Esta propiedad de transformar mediante torsión y distensión se conoce como *Deformación Topológica*.

2.5.1. Espacio topológico

Desde un punto de vista axiomático, una *topología* T de un conjunto X es una familia de subconjuntos de X que satisface:

a) X, $\emptyset \in T$.
b) Si $G_\alpha \in T$, $\forall_\alpha \in \Lambda$ entonces, $(\bigcup G_\alpha : \alpha \in \Lambda) \in T$
c) Si $G_1, G_2, ..., G_n \in T$, entonces $(\bigcap G_i : i = 1, ..., n) \in T$

Si T es una topología para X, entonces los miembros de T son conjuntos abiertos de la topología, y la tupla (X, T) recibe el nombre de espacio topológico (Mansfield, 1964).

2.5.2. Tipos de deformación

En Topología los métodos para demostrar la no-homeomorfia de objetos matemáticos, se pueden clasificar en tres tipos de deformaciones: istópicas, homotópicas y homeomorfismo.

Homeomorfismo

Una biyección bicontinua f de un espacio topológico E sobre otro espacio topológico Y, se llama homeomorfismo; así como su aplicación inversa f^{-1} que es continua y también es un homeomorfismo. Si entre dos espacios topológicos se puede establecer el

homeomorfismo entonces los dos espacios son equivalentes (Satō, 1999). En la Figura 2.5 las curvas $a),b),c)$ son homeomorfas.

Isotopía

Sean $f,g : X \to Y$ mapeos de espacio topológico X, sobre el espacio topológico Y, y si f y g preservan o invierten la orientación, entonces f,g son isotópicas. Al contrario de la isotopía el homeomorfismo depende de las propiedades del objeto en si, propiedades intrínsecas de los objetos; en la Figura 2.5, las curvas $b)$ y $c)$ son homeomorfas más no isotópicas pues el sentido establecido en las dos $b)$ y $c)$ no se corresponden. En las deformaciones isotópicas de dos objetos; X, Y, todas las posiciones intermedias que adopte el objeto X para su transformación en Y deben estar orientadas en un mismo sentido; así, las curvas $a)$ y $b)$ de la Figura 2.5 son isotópicas, mas $a)$ y $d)$ pese a tener el mismo sentido no son isotópicas al tener $d)$ un punto doble (Satō, 1999).

Homotopía

Sean $f,g : X \to Y$ mapeos de espacio topológico X, sobre el espacio topológico Y, f es homotópica a g, si existe un mapeo continuo, $F : X.[0,1] \to Y$, tal que
$F(x,0) = f(x)$, $F(x,1) = g(x)$, $\forall x \in X$.
La deformación homotópica ya no exige que las posiciones intermedias de los objetos que se van a deformar no tengan puntos dobles; a lo largo de la deformación del objeto X en el objeto Y, X puede cortarse arbitrariamente. Entonces estas estructuras en donde podemos aplicar una deformación homotópica se las llama homotopas o *libremente* homotopas. Dentro de esta estructura formal los objetos o curvas isotopas son un subconjunto de las homotopas. La cuatro figuras que se exponen en la Figura 2.5 son homotopas. Isotopía y Homología son conceptos extrínsecos (Satō, 1999).

2.5.3. Resolución de problemas por deformación

El objetivo de la presente sección, es introducir una de las herramientas más simples y poderosas de la Topología; la *deformación topológica*.

Ejemplo 1

¿Es posible conectar con curvas continuas los segmentos AA',DD' y CC' Figura 2.6 $a)$ sin que estas se corten en ningún punto?
A primera vista el problema mostrado en la Figura 2.6 $a)$ resulta imposible de resol-

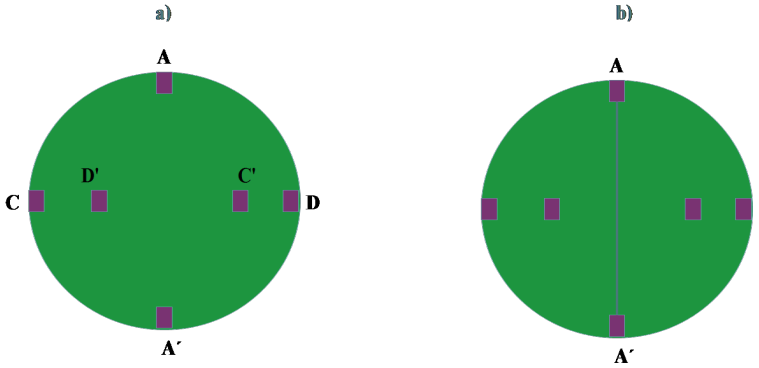


Figura 2.6: a) Problema original, b) Intento de solución

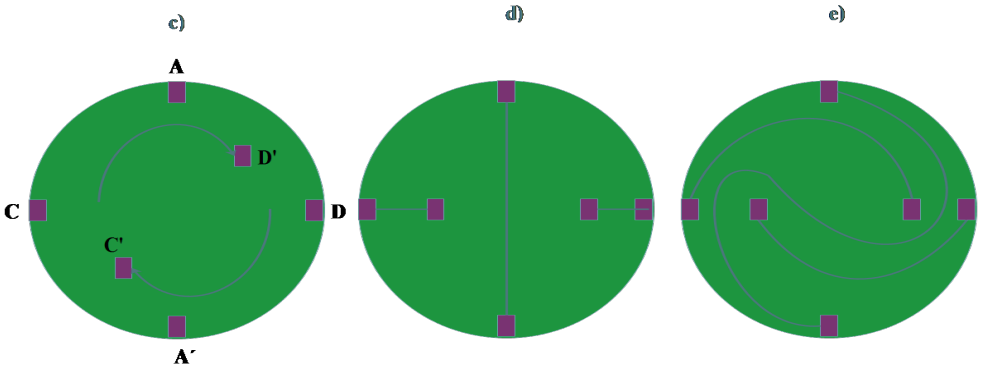


Figura 2.7: Solución del problema por deformación

ver, especialmente si se intenta unir los segmentos AA' como se muestra en la Figura 2.6 b). Ejemplo tomado de (Polya, 2014). Este ejemplo podría parecer insustancial, no obstante es la forma más fácil de entender un concepto tan abstracto dentro de la Topología; la *deformación topológica*. Para solucionar este problema es posible aplicar la *deformación*. Si deformamos el problema original mostrado en la Figura 2.6 a) en un problema fácil de resolver permutando los puntos C' y D' como se muestra en la Figura 2.7 c). El nuevo problema *deformado* Figura 2.7 c) es muy fácil de resolver, como se muestra en d), después se aplica la deformación inversa, que transforma la solución d) en la solución del problema original, como se muestra en e), que resulta ser la solución al problema por *deformación*.

Las soluciones de algunos problemas matemáticos no lineales complejos, en el área de las ecuaciones diferenciales, son inmunes a la deformación (Vladimir I. Arnol'd, 1992) (H. Keith Moffatt (auth.), 2001). El uso de la Topología permite simplificar la complejidad del problema. En matemática los problemas que presentan gran complejidad son los casos especiales o *problemas degenerados* en donde no es posible aplicar un método ortodoxo, desde el punto de vista de la Topología, es posible deformar estos

problemas en problemas genérico fácil de resolver, siguiendo la misma filosofía de la deformación, podemos encontrar una forma de regresar del problema *deformado* al original (Tokieda, 2001).

Ejemplo 2

Sean A, B matrices cuadradas de dimensión $n_x n$, bajo la regla del producto de matrices en general se sabe que $AB \neq BA$; no obstante se sabe que AB y BA tienen los mismos *valores propios*. Demostrar que AB y BA tienen los mismos *valores propios*.
Se debe probar que a partir de AB se puede llegar a BA, entonces si multiplicamos por izquierda por A^{-1} y por la derecha de (AB) por A se tiene que; $A^{-1}(AB)A$ entonces $A^{-1}(AB)A = BA$; por lo tanto (AB) como matriz esta relacionada con (BA) por la multiplicación de una matriz y su inversa. Entonces (AB) y (BA) son matrices similares o desde un punto de vista geométrico (AB) y (BA) son dos matrices para la misma transformación lineal (matrices que representan el mismo objeto geométrico). Por lo tanto (AB) y (BA) tienen los mismos *valores propios*.
Pero esto sólo es verdad si A^{-1} existe; la solución que damos es una solución particular (*deformada*), es valido únicamente si $det(A) \neq 0$. En este contexto es necesario considerar, qué sucede si la matriz A es una matriz degenerada. La situación cuando $det(A) = 0$ es un caso degenerado y cuando $det(A) \neq 0$ es una situación genérica fácil de resolver y sucede en la mayoría de los casos.
Ahora bien, si la matriz (A) es invertible y si se le aplica una perturbación muy pequeña, cambiando las entradas de la matriz, esta seguirá siendo invertible, no obstante si a una matriz no invertible se le practica la misma perturbación, muy probablemente el determinante dejara de ser cero.
Es fácil demostrar esta afirmación si *deformamos* la matriz A cuyo determinante es cero de la siguiente forma:

$$|A| = \begin{vmatrix} 1 & 0 \\ 0 & 0 \end{vmatrix} = \lim_{n\to\infty} \begin{vmatrix} 1 & 0 \\ 0 & \frac{1}{n} \end{vmatrix} \tag{2.7}$$

En conclusión, cuando $n \to \infty$ se cumple que (AB) y (BA) tienen los mismos *valores propios*, ya que, únicamente depende de la continuidad de matrices cuyos determinantes son diferentes de cero. Ejemplo tomado de (Tokieda, 2001).

Ejemplo 3

Demostrar que para toda matriz cuadrada A, $det(e^A)$=$e^{tr(A)}$
Se sabe que si A es diagonalizable entonces $A = S\Lambda S^{-1}$ si tomamos la exponencial a

los dos miembros de esta igualdad tenemos $e^A = Se^{\Lambda}S^{-1}$ entonces los valores $\lambda_1, ...\lambda_n$ *valores propios* del polinomio característico son las entradas de la matriz Λ después de la diagonalización de A, además note que $det(e^A) = Se^{det(\Lambda)}S^{-1}$; lo cual demuestra la igualdad para el caso genérico cuando A es diagonalizable. Sólo las matrices genéricas son diagonalizables. Si a una matriz diagonalizable se la *deforma* introduciendo pequeñas perturbaciones en las entradas; esta continua siendo diagonalizable; lo que no sucede con las matrices singulares, muy pequeñas perturbaciones en las entradas de estas matrices, permiten que estas matrices sean diagonalizables. Ademas una matriz singular es el límite al que tienden las matrices genéricas, lo cual permite introducir perturbaciones del orden infinitesimal. Es muy fácil demostrar esta afirmación si expresamos la matriz singular como el límite de una matriz genérica como en el ejemplo 2. Problemas genéricos resultan fáciles de demostrar y los problemas degenerados se puede demostrar por continuidad. Esta misma filosofía se puede aplicar para demostrar que no es posible resolver ciertos problemas (Tokieda, 2001).

2.6. Variantes del Algoritmo Genético, deformación topológica y reparación

2.6.1. Espacio de búsqueda

En problemas de optimización estamos interesados en buscar soluciones que son mejores que otras, el conjunto de todos estas posibles soluciones se conoce con el nombre de *espacio de búsqueda*, que está compuesto por soluciones factibles y no factibles. En problemas de optimización combinatoria se requiere maximizar o minimizar cierta función objetivo, en la mayoría de los casos el tamaño del espacio de búsqueda es muy grande, por lo que no se puede saber a priori dónde empezar la búsqueda y más aún no es posible saber si el valor obtenido es el valor óptimo véase Figura 2.8, la única forma de asegurar que la solución encontrada representa el valor óptimo; sería explorar todo el espacio de búsqueda, de alguna forma, lo cual en la mayoría de los casos resulta inaplicable debido al requerimiento computacional.

Las aplicaciones del algoritmo genético desde su creación hasta la actualidad, tienen una gran envergadura, el algoritmo genético ha demostrando una eficiencia y una eficaz exploración del *espacio de búsqueda*, esto es discutido a profundad en los trabajos de (Michalewicz y Hartley, 1996), (Luke, 2013), (Rajasekaran y Pai, 2003) y (Goldberg, 2002)

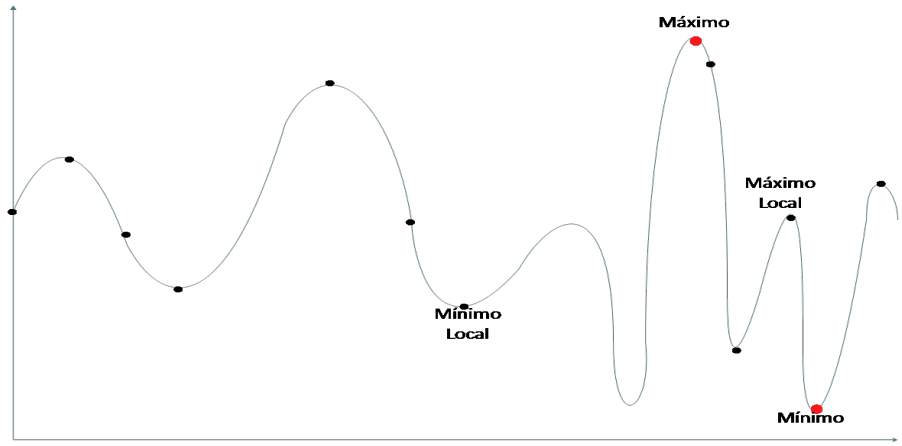


Figura 2.8: Ejemplo de un espacio de búsqueda

2.6.2. Reparación

Una variante del algoritmo genético canónico es la de introducir una función de reparación. Cuando se aplica el operador de cruzamiento (Crossover) es inevitable la creación de cromosomas que representan soluciones no factibles. Para lo cual se establece un proceso de reparación, en el caso del *KP* las soluciones no factibles se dan cuando éstas violan la capacidad del «Knapsack», en este caso particular se suele excluir elementos del «Knapsack» hasta que se encuentre dentro del área de factibilidad (Rashid, 2010).

2.6.3. Función de penalización

La función de penalización es una alternativa a la reparación, esta penaliza el fitness de los individuos para evitar que estos se reproduzcan, cada vez que se evaluá la calidad del cromosoma; se verifica que este cromosoma no viole las restricciones del problema que tratamos de solucionar. Si el cromosoma evaluado representa una solución no factible, su valor de fitness es penalizado. En el caso particular del *KP* donde la solución se representa como una secuencia de bits de longitud n, donde si un objeto es seleccionado para ser colocado en la mochila el gen en dicha posición es marcado como $x_{[i]} = 1$ entonces el valor del fitness para cada cromosoma es asignado como :

$$eval(x) = \sum_{i=1}^{n} x_{[i]}.p_{[i]} - pen(x)$$

donde la función *pen(x)* es cero para las soluciones factibles, aquellas donde se verifica que $\Sigma_{i=1}^{n} x_{[i]}.w_{[i]} \leq C$, caso contrario, la función de penalización toma un valor, dependiendo del tipo de penalización que se aplica.

La penalización puede ser de tipo: logarítmica, lineal y exponencial como se expone a continuación

- $pen(x) = \rho(\sum_{i=1}^{n} x_{[i]} w_{[i]} - C)$, penalización lineal.
- $pen(x) = (\rho(\sum_{i=1}^{n} x_{[i]} w_{[i]} - C))^2$, penalización cuadrática.
  donde $\rho = max_{i=1,\ldots n}(p_{[i]}/w_{[i]})$

2.6.4. Deformación

El Algoritmo Genético al ser una metaheurística basado en población, se recomienda que esta tenga diversidad de individuos; que amplíen el espacio de búsqueda, y evitar la convergencia prematura (Luke, 2013), además es posible emplear alguna heurística para generar esta población inicial, el único objetivo de usar una heurística en la generación de la población inicial es integrar buenos genes a la población (Affenzeller et al., 2009). Una alternativa para agregar buenos genes a la población del Algoritmo Genético, es aplicar un algoritmo conocido que de una buena solución local, como: búsqueda tabú, búsqueda local, «Simulated annealing», etc.. La desventaja de emplear un algoritmo, previa la ejecución del Algoritmo Genético, radica en el rendimiento, además de correr el riesgo de restringir la población a un área única lo que podría llevar a una convergencia prematura. La *Deformación topológica* podría brindar buenos genes a la población inicial, si partimos de soluciones que se encuentren en el área de no factibilidad y las sometemos a perturbaciones, esto podría converger en un buen individuo con un alto valor de *fitness*. Además, el límite al que converge la *deformación* de una solución no factible, es una solución factible (Luke, 2013). Esta deformación es diferente al operador de mutación del algoritmo genético y no debería dar lugar a confusiones. El operador de mutación, se aplica a todos los genes del cromosoma, con una probabilidad muy baja, con el objetivo de recuperar material genético que se pudo perder al aplicar el operador *crossover* en la reproducción (Rashid, 2010). El algoritmo 1 expone la deformación.

Al ingresar estas perturbaciones, a las soluciones no factibles que se encuentran en las áreas con un *fitness* muy alto, eventualmente el cromosoma resultante podría salir del área de no factibilidad con un valor de *fitness* muy alto, el número de individuos generados con este tipo de heurística debería ser muy bajo para garantizar la diversidad de la población, por lo que los individuos de la población generada de forma aleatoria debería ser mucho mayor a los individuos que fueron generados usando la deformación.

Algoritmo 1 Operador de Deformación

//Genera una solución no factible con un altísimo fitness
$X \leftarrow$ solución-no-factible
// Generar un número aleatorio (número de deformaciones)
$r_1 \in [1,...n/2]$
mientras $r_1 > 0$ hacer
// Generar un número aleatorio (índice de deformación)
$r_2 \in [1,...n]$
$X_{r_2} \leftarrow$ null
$r_1 - -$
fin mientras
retorna X
Fin

2.7. Algoritmo Ramificación y Poda

El algoritmo Ramificación y poda (del inglés «Branch and Bound» (B&B)) aplicado al *KP* fue introducido por vez primera en (Kolesar, 1967), este algoritmo explora todo el *espacio de búsqueda* de una manera *inteligente*. El algoritmo garantiza que en ciertas ramas ya no es posible encontrar soluciones óptimas y procede a podar (Kellerer et al., 2004), limitando el número de nodos abiertos en memoria a $O(n)$. El algoritmo requiere que previamente se ordenen los objetos en función de $\frac{Beneficio}{Volumen}$, este ordenamiento previo tiene complejidad $O(n \log n)$. Suponiendo que se requiere maximizar la función objetivo $\underset{x \in X}{Max} f(x)$ donde X representa el espacio de búsqueda que es finito; y sea $X' \subseteq X$ un subconjunto de soluciones del espacio X, las cuales se dividen en subconjuntos más pequeños $X_1, X_2, ...X_m$ que podrían estar sobrepuestas, pero la unión de todas deben generar el universo de soluciones X (Winston y Goldberg, 2005), como $\bigcup_{i=1}^{m} X_i = X'$, este proceso se repite hasta que cada subconjunto represente una solución factible, el mejor valor de estos subconjuntos es el valor del **óptimo global**.

El proceso de poda del **(B&B)** se basa en dos limites; superior e inferior $z^i \leq z^*$, el límite superior (del inglés Upper Bound) $U_{X'}$ de una solución particular tal que $X' \subseteq X$ es un número real tal que:

$$U_{X'} \geq f(x) \quad \forall x \in X' \tag{2.8}$$

este límite superior es usado como referencia para hacer la poda. Supongamos que $U_{X'} \leq z^i$ para algún subconjunto de X', entonces de la Inecuación 2.8 se tiene que:

$$f(x) \leq U_{X'} \leq z^i \quad \forall x \in X' \tag{2.9}$$

la desigualdad mostrada en la ecuacion 2.9 significa que no es posible encontrar una mejor solución en el subconjunto $X^{'}$ por lo tanto, no es necesario ramificar (Kolesar, 1967).

El algoritmo 2 presenta **B&B** de forma recursiva aplicado al *KP*.

Algoritmo 2 BranchBound(i) (Kolesar, 1967)

Inicio

i es el índice del nuevo objeto a ser agregada a la solución actual tal que

$X_j = X_j^{'}$ *para* $j = 1,...,i-1$

El espacio de la solución actual es

$$X^{'} = \{x_j \in \{0,1\} | \sum_{j=1}^{i-1} w_j x_j^{'} + \sum_{j=i}^{n} w_j x_j \leq c\}$$

si $\sum_{j=i}^{i-1} w_j x_j^{'} > c$ retorna *Cuando* $X^{'}$ *esta vacía.*

si $\sum_{j=i}^{i-1} p_j x_j^{'} > z$ hacer

$z = \sum_{j=1}^{i-1} x_j, x_j^{*} = x^{'}$ *Mejorar la solución*

si $(i > n)$ retorna $X^{'}$ *contiene el óptimo global*

Obtener $U_{X'} = max_{x \in Y} f(x)$ *límite superior*

donde $Y = \{x_j \in [0,1] | \sum_{j=1}^{i-1} w_j x_j^{'} + \sum_{j=i}^{n} w_j x_j \leq c\}$

si $u_{X'} > z$ hacer

$x_i^{'} = 1, Branch - and - Bound(i+1)$

$x_i^{'} = 0, Branch - and - Bound(i+1)$

Fin

Capítulo 3

Materiales y métodos

3.1. Región de estudio

Para el presente estudio se utilizó una parte de la cuenca del Río Tabacay, que a su vez es una subcuenca del Río Paute mostrada en la Figura 3.1. El área de la cuenca del Río Tabacay es de aproximadamente 66.3 km^2, y su altitud fluctúa entre 2481 y 3732 msnm. El Río Tabacay contribuye al abastecimiento de agua potable a la ciudad de Azogues además de servir como fuente para riego en la agricultura (Estrella et al., 2014).

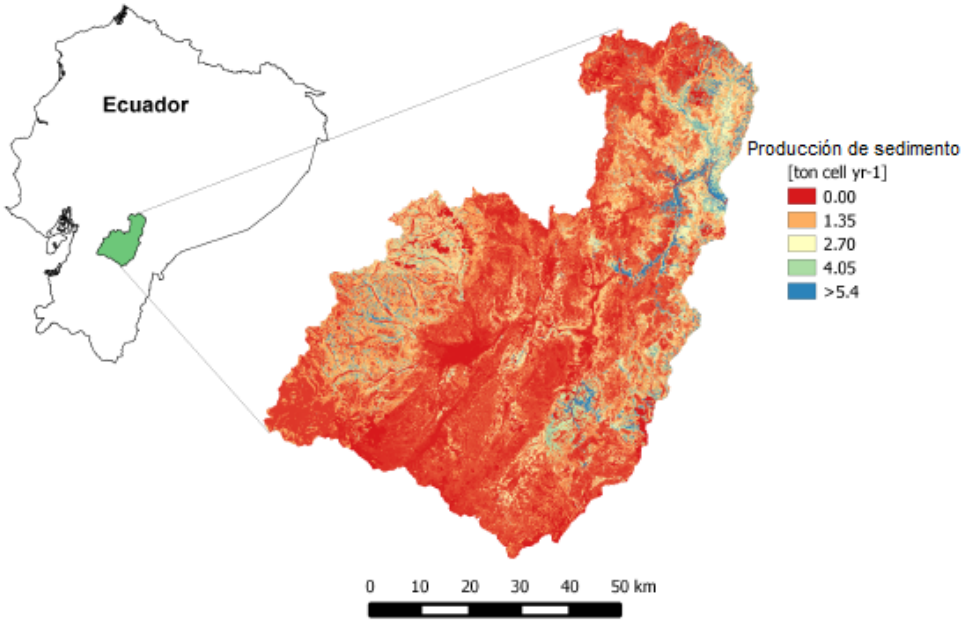


Figura 3.1: Cuenca del Río Paute (Estrella et al., 2014)

3.2. Implementación de CAMF

Para el presente estudio se implementó el algoritmo *CAMF* basado en el código fuente desarrollado por (Estrella et al., 2014) para lo cual se pidió autorización al autor. El algoritmo *CAMF* está basado estrictamente según las especificaciones de (Vanegas Peralta et al., 2010) en el lenguaje de programación *Java*. En este proyecto se requiere aplicar el algoritmo *CAMF* para asignar los beneficios de forestación (reducción de sedimento) a las áreas representadas en el mapa ráster (matriz de datos).

Datos de entrada para los métodos utilizados

Se utiliza una parte de la cuenca, correspondiente a un área total de $35km^2$ que corresponde al 53 % del área total de la cuenca dividida en 39.688 celdas con una resolución de 30x30 m en formato ArciInfo ASCII grid como se indica en la Figra 3.2. Para generar la lista de celdas candidatas a ser forestadas, con su respectivo benéfico se aplico el algoritmo **CAMF**. La salida de este algoritmo es la lista con las celdas candidatas a ser forestadas con su respectivo beneficio de forestación.

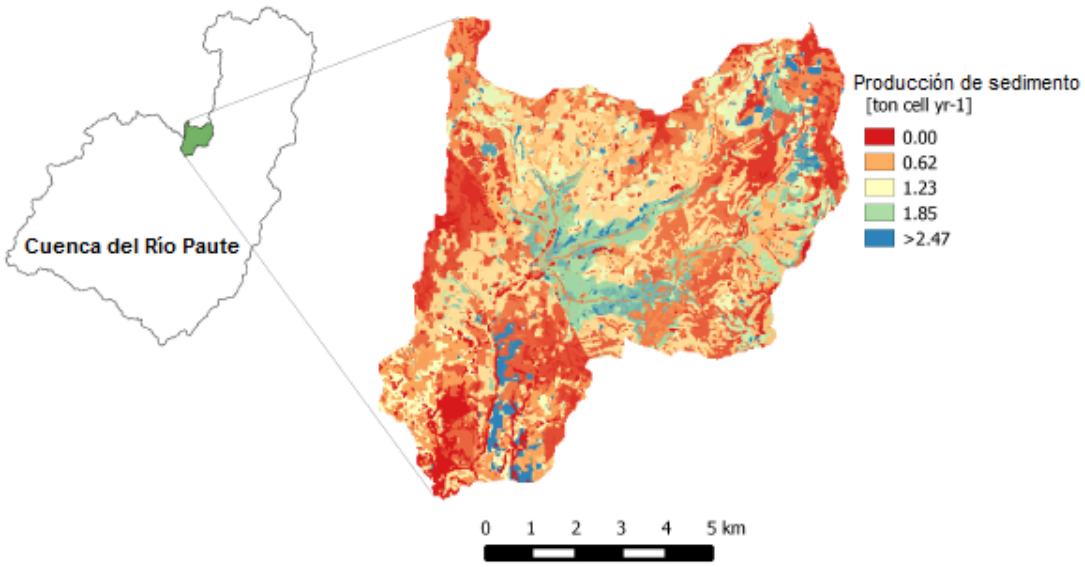


Figura 3.2: Cuenca del Río Tabacay (Estrella et al., 2014)

El algoritmo *CAMF* para su ejecución requiere de los siguientes datos:

1. Un mapa ráster con la producción de sedimento: cada celda produce sedimento expresada en $kg/km^2\ yr^{-1}$

2. Capacidad de retención: cada celda tiene asociada una capacidad de retención expresada en $kg/km^2\ yr^{-1}$, si la producción de sedimento de la celda es menor que la capacidad de retención entonces esta celda no desplaza sedimento a ninguna celda vecina.

3. Umbral de saturación: toda celda cuya producción de sedimento sobrepase el nivel de saturación tiene que desplazar sedimento expresado en $kg/km^2\ yr^{-1}$ a la celda con más pendiente de inclinación hacia abajo.

4. Factor de flujo: este *dataset* indica la fracción de sedimento que se va a desplazar de la celda hacia la celda vecina. Este factor se aplica únicamente cuando la producción de sedimento se encuentra en el intervalo entre la capacidad de retención y el umbral de saturación

5. Mapa ráster con dirección de flujo en una sola dirección (del inglés Single Flow Direction raster map) de la cuenca en cuestión, indica el camino que tiene que seguir el sedimento hasta la salida de la cuenca.

6. Número de celdas: este parámetro indica el número de celdas que debe contener la solución, el método **CAMF** construirá una lista de este tamaño con las mejores celdas candidatas a ser forestadas.

CAMF-B

El método de solución heurística basado en autómatas celulares para minimizar el flujo sujeto a restricción presupuestario (del inglés CA-Based Heuristic Method for Minimizing Flow Under a Budget Constraint CAMF-B) introducido por (Vanegas et al., 2010) para restringir la selección de celdas que serán forestadas en función de un presupuesto fijo es una variante de **CAMF**, que considera la restricción del presupuesto para seleccionar las celdas a ser forestadas, en donde no emplea un número fijo de celdas; e integra estrategias del «**Knapsack Problem**». Combinando estas dos estrategias, cada celda está asociada a un costo de forestación, que se mide, en unidad de celda y un beneficio (reducción de sedimento en la salida de la cuenca) que está dada por la aplicación de CAMF. Para resolver el «**Knapsack**» resultante de aplicar **CAMF-B** se optó por emplear Sahni's PTAS algorithm (Sahni, 1975), por tres motivos 1) facilidad de implementación y combinación con CAMF; 2) se puede parametrizar la calidad de la solución y 3) el objetivo fundamental en este caso de estudio, fue resolver el problema de forestación en un tiempo corto y con soluciones de calidad. **CAMF-B** se probó en problemas de forestación pequeños y Sahni's PTAS algorithm (Sahni, 1975) integrado

tiene complejidad $O(n^{k+1})$ (Vanegas et al., 2010), lo cual demanda gran cantidad de recursos para problemas de forestación donde se tenga gran cantidad de celdas debido a la resolución que se esté aplicando. Por esta razón en este estudio no se utilizó el algoritmo *CAMF-B*.

Características de los equipos donde se probaron los algoritmos

Características del equipo	
Procesador:	Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz
Número de núcleos:	2 (max 8)
Memoria RAM:	8 GB
Sistema operativo:	Windows 8, 64 bits

Tabla 3.1: Características de la CPU, memoria y sistema operativo

3.3. Asignación de beneficio a las áreas por forestar mediante CAMF

El algoritmo *CAMF* en nuestro caso de estudio específico, únicamente construye la lista con los benéficos de forestación (reducción de sedimento) de cada una de las celdas candidatas. En este punto el algoritmo se detiene, luego con estos datos se procede a integrar estrategias del *KP* asignando a cada celda candidata el costo de forestación respectivo como distancias a vías y se aplican los algoritmos *Branch and Bound* y el *Algoritmo Genético* con sus variantes.
El algoritmo *CAMF* se ejecutó con 39.688 celdas, produciendo un ranking de 21.669 celdas candidatas a ser forestadas, que representan las mejores áreas; de esta lista se detectó áreas que no contribuyen, es decir celdas con beneficios de cero, éstas áreas son aquellas que están fuera de la cuenca o a su vez si están dentro de la cuenca, no es posible forestar por ser cercanas a los ríos, o representar alguna vía de acceso etc. Por lo que se debe filtrar estas celdas de la lista, ya que mientras más celdas tienen el problema de forestación, el «Knapsack» resultante resulta más complejo. Una vez filtrado las celdas que no contribuyen en nada a la reducción de sedimento se reduce a 15.014 celdas candidatas.

3.4. Determinación de costo de forestación basado en distancias a vías

Para lograr construir la matriz con los costos de forestación se utilizó un mapa de vías de la región de estudio en formato ArciInfo ASCII grid con una resolución de 30x30 m. El costo de forestación de una celda equivale a la distancia entre dicha celda y la vía más próxima: para la celda (j) se busca en su vecindad la celda más cercana que represente una vía y se le asigna esta distancia como costo de forestación. Es posible que exista otras formas de asignar el costo de forestación a las áreas en el mapa ráster, no obstante abordar una metodología para asignar costos de forestación a las celda ésta fuera del alcance de estudio. Para asignar los costos de forestación se implementó un algoritmo en el lenguaje de programación *java*.

3.5. El problema de selección de áreas óptimas para forestación modelado como un «Knapsack Problem»

Esta Sección está basada en el Capítulo 6 *Locating Sites in Raster Maps for Minimal Off-site Impact Under a Budget Restriction* **CAMF-B** introducido por (Vanegas et al., 2010).

El algoritmo **CAMF** construye una lista con el ranking de las celdas candidatas a ser forestadas y asignado los costos de forestación en función de las distancias a vías. Al tener dos objetivos concurrentes; maximizar el beneficio (disminución de la producción de sedimento) sin exceder el presupuesto establecido, permite modelar el problema de forestación como un «Knapsack Problem»; los objetos estarían representados por celdas ráster, el beneficio correspondería a la minimización de la producción de sedimento, mientras que la capacidad del «Knapsack» estaría representada por el presupuesto disponible en unidades de celda. Sea C el presupuesto de forestación dado en unidades de celda, cada celda j está asociado a un beneficio p_j, y un costo w_j. Se requiere maximizar la suma de los beneficios de las celdas elegidas para ser forestadas, de tal forma que la suma de los costos asociados a las celdas no exceda la capacidad el presupuesto establecido. Formalmente definido por (3.1):

$$\begin{aligned} &\textit{Maximizar} \quad \sum_{j=1}^{n} p_j x_j \\ &\textit{Sujeto a:} \\ &\sum_{j=1}^{n} w_j x_j \leq \mathrm{C} \quad x_j \in \{0,1\} \forall j \quad , j \in \mathrm{N} \end{aligned} \tag{3.1}$$

En base a esta filosofía se han construido siete problemas de forestación, donde se probarán los algoritmos, con diferentes números de celdas y presupuestos diferentes que se exponen en la Tabla 3.2, también se probará con presupuestos iguales como se expone en la Tabla 3.3.
Para asignar el presupuesto a los problemas de forestación, se tomó en consideración el presupuesto requerido para forestar todas las celdas de cada problema, y considerando que el *KP* requiere que este valor sea menor, se optó por asignar la mitad de este valor. Existen metodologías especializadas, en asignar este presupuesto, en el área de análisis de costos, no obstante este proceso esta fuera del alcance de este proyecto.

Problemas de forestación	
Número de celdas	Presupuesto en UC(unidades celda)
1.000	2,039
2.000	4,268
3.000	6,089
4.000	7,802
7.000	12,233
10.000	18,290
15.014	24,842

Tabla 3.2: En cada caso el espacio de búsqueda es de 2^n donde n es el número de celdas.

3.6. Implementación del Algoritmo Genético

Para implementar el *Algoritmo Genético* se utilizó el lenguaje de programación *java* y librerías especializadas del framework *JGAP* versión 3.4.4 para *Algoritmos Genéticos*. La implementación y el código fuente se exponen el Apéndice A

Problemas de forestación con presupuestos iguales	
Número de celdas	Presupuesto en UC(unidades celda)
1.000	2,039
2.000	2,039
3.000	2,039
4.000	2,039
7.000	2,039
10.000	2,039
15.014	2,039

Tabla 3.3: En cada caso el espacio de búsqueda es de 2^n donde n es el número de celdas.

Codificación de los genes de la población del Algoritmo Genético para el problema de forestación

Existe muchas formas de codificar los genes que conformaran los cromosomas del *Algoritmo Genético* (Rashid, 2010), en esta investigación se optó por la codificación booleana, el cual es compatible con el problema de forestación modelado «Knapsack Problem» binario. Entonces la cadena del **ADN** está constituida por genes cuyos *alleles* son valores de falso o verdadero, como se muestra en la Figura 3.3. Para configurar los genes se utilizó la clase *Gen* de *JGAP*. Con esta configuración podemos identificar la secuencia única de celdas que deben ser forestadas luego de que el *Algoritmo Genético* termine la ejecución. Una posible solución es una secuencia de genes como se muestra en la Figura 3.3, donde se puede identificar las celdas que fueron seleccionadas para ser forestadas.

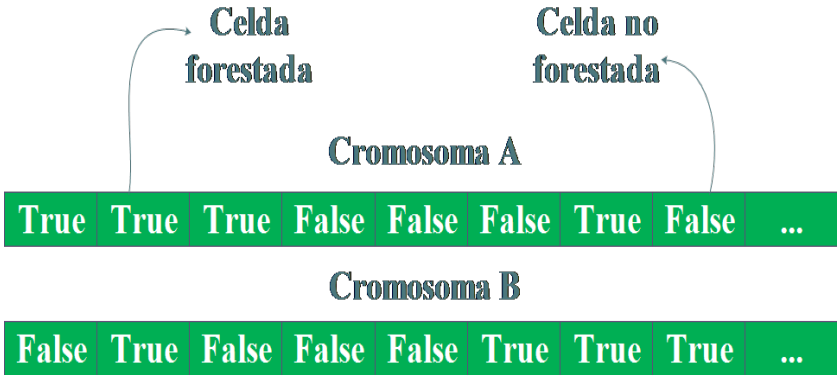


Figura 3.3: Codificación de las soluciones candidatas en el *ADN* de los individuos que conformaran al población del *Algoritmo Genético*.

Codificación de la población del Algoritmo Genético

En los problemas de forestación expuestos en la Tabla 3.2 el tamaño de la población es un parámetro variable, que depende de la cantidad de tiempo que estemos dispuestos a invertir y la cantidad de memoria del ordenador, en los problemas de forestación donde intervienen gran cantidad de celdas, debido a las limitaciones de memoria, se optó por establecer en 500 individuos. Para construir la población se utilizó la clase *Genotype* del paquete *JGAP*, el cual va ha contener a todos los individuos de la población, véase Apéndice A.

Función objetivo

El *Algoritmo Genético* requiere de una función objetivo, para el problema de forestación, se implementó la función objetivo especificada en la Ecuación 3.1; esto se logra extendiendo la clase *FitnessFunction* de *JGAP* y se debe sobreescribir (Override) véase Apéndice A, la función *eval*, integra las estrategias de la función objetivo del problema de forestación.

Función de penalización

Para el presente estudio se optó por utilizar penalización lineal en lugar de usar la reparación, el proceso de reparación en las pruebas realizadas requiere un número elevado de recursos, cuando la longitud de cromosomas es grande, por lo cual no se utilizó la reparación véase Apéndice A. En su lugar se aplica la penalización lineal que cumple el mismo objetivo que la reparación. Cuando un cromosoma representa una solución no factible, se procede a penalizar su valor de *fitness*, este valor está asignado con la función *eval(x)* donde x representa un cromosoma arbitrario. La penalización es cero, para cromosomas que representan soluciones factibles, esto es, si cumple $\sum_{i=1}^{n} w_{[i]}x_{[i]} \leq C$, caso contrario se penaliza de acuerdo con la Ecuación 3.2:

$$
\begin{aligned}
&eval(x) = \sum_{i=1}^{n} p_{[i]}x_{[i]} - pen(x) \\
&donde \\
&pen(x) = \rho(\sum_{i=1}^{n} w_{[i]}x_{[i]} - C) \qquad x_{[i]} \in \{0,1\}\forall i \\
&y \\
&\rho = max_{i=1,\ldots n}(p_{[i]}/w_{[i]})
\end{aligned}
\tag{3.2}
$$

$$eval(x) = \sum_{i=1}^{n} x_{[i]}.b_{[i]} - pen(x)$$

Función de deformación

La deformación se aplica a un porcentaje muy reducido de la población. En este contexto se necesita que la población inicial tenga buenos genes, pero también que sea diversa, en esta investigación se optó por aplicar como la heurística basada en la *deformación topológica* al 10% de la población total; esto implica que el 10% de individuos que conforman la población inicial, son generados de forma artificial, tratando de aportar buenos cromosomas, el resto de la población se deja al azar. Para generar los individuos de la población se utiliza la función *randomInitialGenotype* de la clase *Genotype* de la libreria JGAP, que permite generar la población de forma aleatoria. El operador de deformación se expone en el Algoritmo 1 y véase Apéndice A.

3.7. Implementación del algoritmo «Branch and Bound»

Para implementar el algoritmo se utilizó el lenguaje de programación *java*. El algoritmo está basado en el algoritmo versión recursiva expuesto en el Algoritmo 2. La salida de este algoritmo es el valor del óptimo global de los problemas de forestación, junto con la lista de celdas que deben ser forestadas. La implementación y el código fuente se exponen el Apéndice B

Capítulo 4

Resultados y discusión

4.1. Algoritmo «Branch and Bound»

En la Tabla 4.1 se resumen los resultados obtenidos con *Branch and Bound*. El algoritmo es capaz de encontrar el óptimo global del problema de 10.000 celdas en un tiempo de 64s, lo cual, es un resultado inesperado, dado que «Knapsacks» de hasta 500 celdas, podrían requerir, varios minutos en converger al valor óptimo, no obstante se evidencia que «Knapsacks» construidos aplicando el algoritmo **CAMF** para asignar los beneficios (reducción de sedimento), a las celdas que serán forestadas, y dado que **CAMF** entrega la lista de celdas candidatas ordenadas, el algoritmo *Branch and Bound* es capaz de encontrar un límite inferior muy rápido que se ve reflejado en el tiempo de convergencia hacia el óptimo global como se muestra en la columna 5 de la Tabla 4.1. Es importante notar que *Branch and Bound* calculó el valor del óptimo global, para el problema de 10.000 celdas, no obstante se volvió prohibitivo, cuando se cambió el presupuesto de este problema a 2.039, lo cual implica que algunos «Knapsacks», al tener un número excesivo de celdas y un presupuesto extremadamente reducido, el algoritmo puede ser inaplicable, como se muestra en la Tabla 4.2. Esto se debe a que el presupuesto es extremadamente reducido, por lo que muy pocas celdas serán parte de la solución, esto dificulta encontrar un límite inferior, que permita al algoritmo podar nodos que se abren, llegando a saturar completamente la memoria. Otro resultado inesperado se da en el problema de 7.000 celdas cuando reducimos el presupuesto a 2.039, el tiempo en converger es mucho menor que para el caso de 4.000 celdas con el mismo presupuesto como se ve en la Tabla 4.2, esto se debe a que cada «Knapsack» es un problema diferente y el tiempo de ejecución depende de encontrar el límite inferior y este valor está en relación directa con el presupuesto y el número de celdas del problema, esto se ve reflejado en los tiempos, que son mucho mayores en los problemas de forestación donde

se redujo el presupuesto a un único valor como se ve en la Tabla 4.2, comparado con los tiempos de los problemas donde se asignó la mitad del presupuesto requerido, para forestar todas las celdas como se muestra en la Tabla 4.1.

Soluciones óptimas, aplicando *Branch and Bound*
Presupuestos distintos

Problemas de forestación		**Resultados**		
Número de Celdas	Capacidad (Uc)	Óptimo Global Beneficio	Costo (Capacidad utilizada)	Tiempo en segundos
1000	2,039	$791,71\ kg/km^2\ yr^{-1}$	2,039	$2s$
2000	4,268	$6,071,1\ kg/km^2\ yr^{-1}$	4,268	$4s$
3000	6,089	$42,085,1\ kg/km^2\ yr^{-1}$	6089	$8s$
4000	7,802	$2,04\ t/ha\ yr^{-1}$	7,802	$20s$
7000	12,233	$14,02\ t/ha\ yr^{-1}$	12,233	$34s$
10.000	18,290	$31,17\ t/ha\ yr^{-1}$	18,290	$64s$
15014	24,842	—	—	—

Tabla 4.1: Resultados de *Branch and Bound*

Soluciones óptimas, aplicando *Branch and Bound*
Presupuestos iguales

Número de Celdas	Capacidad (Uc)	Óptimo Global Beneficio	Costo (Capacidad utilizada)	Tiempo en segundos
1000	2,039	$791,71\ kg/km^2\ yr^{-1}$	2,039	$2s$
2000	2,039	$4,671,56\ kg/km^2\ yr^{-1}$	2,039	$8s$
3000	2,039	$33,649,2\ kg/km^2\ yr^{-1}$	2,039	$21s$
4000	2,039	$1,56\ t/ha\ yr^{-1}$	2,039	$102s$
7000	2,039	$7,04\ t/ha\ yr^{-1}$	2,039	$75s$
10.000	2,039	—	—	—
15014	2,039	—	—	—

Tabla 4.2: Resultados de *Branch and Bound*

4.2. Aplicación de una variante del Algoritmo Genético y comparación de resultados con respecto a *Branch and Bound*

Para los problemas de forestación representado como un «Knapsack Problem» se configuró los siguientes parámetros en el algoritmo genético.

- Función de penalización lineal: esto evita la que soluciones no factibles puedan ser elegidas para reproducción.
- Probabilidad de mutación de 0.05: permite diversificar el espacio de búsqueda.
- Número de generaciones 190: indica el número de veces que la población evoluciona y cuándo el algoritmo termina.
- Tamaño de la población de 500 individuos para problemas de hasta 4.000 celdas y 300 para los problemas más grandes: este parámetro indica el número de individuos que tendrá la población.
- Operador de deformación, se aplica al 10 % de los individuos en la población: este parámetro indica el porcentaje de individuos de la población que serán creados artificialmente.

Se puede observar en la Tabla 4.3 que los tiempos de ejecución del *Algoritmo Genético* son mucho mayores a los del algoritmo *Branch and Bound*, esto se da porque la longitud de los cromosomas es igual al número de celdas que intervienen en cada «Knapsack», y el tamaño de la población es relativamente grande, 500 individuos para los problemas de hasta 4.000 celdas, y 300 para los problemas más grandes. El *Algoritmo Genético* sin embargo es capaz de calcular una solución factible, para el problema de forestación de 15.014 lo que no es posible con *Branch and Bound*, por falta de memoria.

Los resultados obtenidos aplicando la *deformación topológica* como heurística, muestran cambios muy pequeños en el beneficio de las nuevas soluciones, como se ve en la tercera columna de la Tabla 4.4, no obstante el tiempo de ejecución se ve incrementado, esto se da por el proceso adicional que debe hacer el *Algoritmo Genético* para generar los individuos artificiales. No se puede aplicar porcentajes superiores al 10 % de la población para la *deformación topológica* ya que de hacerlo, no se garantizaría la diversidad de la población que requiere el *Algoritmo Genético*, y el peligro que la población se restrinja a un área específica del despacio de búsqueda, el único objetivo de la deformación es introducir buenas estimaciones, que luego evolucionará con el

Soluciones aproximadas, aplicando *Algoritmo Genético*				
Problemas de forestación		**Resultados**		
Número de Celdas	Capacidad (Uc)	Aproximación Beneficio	Costo (Capacidad utilizada)	Tiempo en segundos
1000	2,039	$670,39\ kg/km^2\ yr^{-1}$	2,039	68s
2000	4,268	$4{,}635,58\ kg/km^2\ yr^{-1}$	4,266	192s
3000	6,089	$32{,}651,21\ kg/km^2\ yr^{-1}$	6086	317s
4000	7,802	$1,49\ t/ha\ yr^{-1}$	7,801	428s
7000	12,233	$8,97\ t/ha\ yr^{-1}$	12,228	473s
10.000	18,290	$20,81\ t/ha\ yr^{-1}$	18,290	809s
15014	24,842	$187\ t/ha\ yr^{-1}$	24,835	1467s

Tabla 4.3: Resultados de *Algoritmo Genético*

paso de las generaciones, las aproximaciones aplicando el algoritmo genético para el peor de los caso es del 64 % para el problema de 10.000 celdas, lo cual es una buena aproximación considerando las limitaciones de memoria del equipo, tamaño del *espacio de búsqueda*, tiempo y al tratarse de un problema del tipo *NP-Hard*.

Soluciones aproximadas, aplicando *Algoritmo Genético* Deformación topológica del 10 % de la población				
Problemas de forestación		**Resultados**		
Número de Celdas	Capacidad (Uc)	Óptimo Global Beneficio	Costo (Capacidad utilizada)	Tiempo en segundos
1000	2,039	$675,209\ kg/km^2\ yr^{-1}$	2,039	84s
2.000	4,268	$4{,}731,69\ kg/km^2\ yr^{-1}$	4,268	194s
3.000	6,089	$32{,}748,59\ kg/km^2\ yr^{-1}$	6,088	318s
4.000	7,802	$1,53\ t/ha\ yr^{-1}$	7,801	430s
7.000	12,233	$9,01\ t/ha\ yr^{-1}$	12,230	475s
10.000	18,290	$20,93\ t/ha\ yr^{-1}$	18,290	819s
15.014	24,842	$189\ t/ha\ yr^{-1}$	24,842	1490s

Tabla 4.4: Resultados de *Algoritmo Genético*

Para los problemas de forestación donde se asignó un mismo presupuesto, el *Algoritmo Genético* presenta soluciones no factibles, como se muestra en la Tabla 4.5, esto es normal ya que el algoritmo *Algoritmo Genético* trata de maximizar la función objetivo, como se muestra en las Figuras 4.74.5 mediante la combinación del material genético, el algoritmo siempre va ha tratar de maximizar la función objetivo. Al reducir el presupuesto, implica que pocas celdas formaran parte de la solución, al ser la longi-

tud del cromosoma igual al número de celdas, los cromosomas al combinarse tienen probabilidad alta de seleccionar más celdas, que maximizan la función objetivo, pero violan las restricciones del problema.

En los problemas de más de 2.000 celdas, se puede observar en la cuarta columna de la Tabla 4.5 que el costo (capacidad utilizada) es mucho mayor al establecido de 2.039, esto se podría evitar incrementando el número de evoluciones, lo cual implica más tiempo en converger, el resultado es similar para los «Knapsacks» de mayor tamaño por lo se optó por no calcularlos, considerando el tiempo que toma calcular estas soluciones no factibles.

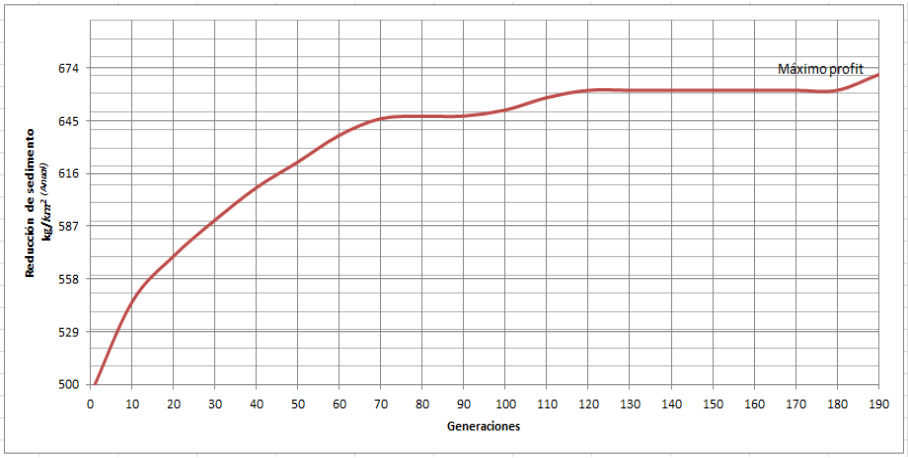
Estos resultados muestran que el *Algoritmo Genético* podría ser prohibitivo para «Knapsacks» donde se tiene un presupuesto extremadamente reducido, en relación al presupuesto total de forestar cada una de las celdas. Además, a estos problemas no se puede aplicar la *deformación topológica* ya que contribuiría en el incremento de soluciones no factibles.

Ahora bien, en problemas de forestación, la probabilidad de tener un presupuesto extremadamente reducido, en relación al presupuesto que se requiera para forestar todas las celdas, es muy improbable que pueda suceder, además, si se trata de minimizar la producción de sedimento mediante la forestación, para que el proyecto tenga un impacto mínimo, se debería considerar un presupuesto que garantice la forestación de un número mínimo de celdas, el presupuesto de 2.039 representa para el problema de 10.000 celdas, el 5.5 % del presupuesto total necesario para forestar todas las celdas de este problema.

Soluciones aproximadas, aplicando *Algoritmo Genético*				
Problemas con presupuestos iguales				
Número de Celdas	Capacidad (Uc)	Óptimo Global Beneficio	Costo (Capacidad utilizada)	Tiempo en segundos
1.000	2,039	657,64 $kg/km^2\ yr^{-1}$	2,039	55s
2.000	2,039	3,419,58 $kg/km^2\ yr^{-1}$	4,232	195s
3.000	2,039	32,651,21 $kg/km^2\ yr^{-1}$	6,086	317s
4.000	2,039	1,02 $t/km^2\ yr^{-1}$	7,747	445s
7.000	2,039	7,29 $t/km^2\ yr^{-1}$	12,472	467s

Tabla 4.5: Resultados de *Algoritmo Genético* problemas de forestación el mismo presupuesto

Figura 4.1: Algoritmo Genético, 1.000 celdas candidatas, Generaciones vs Beneficio



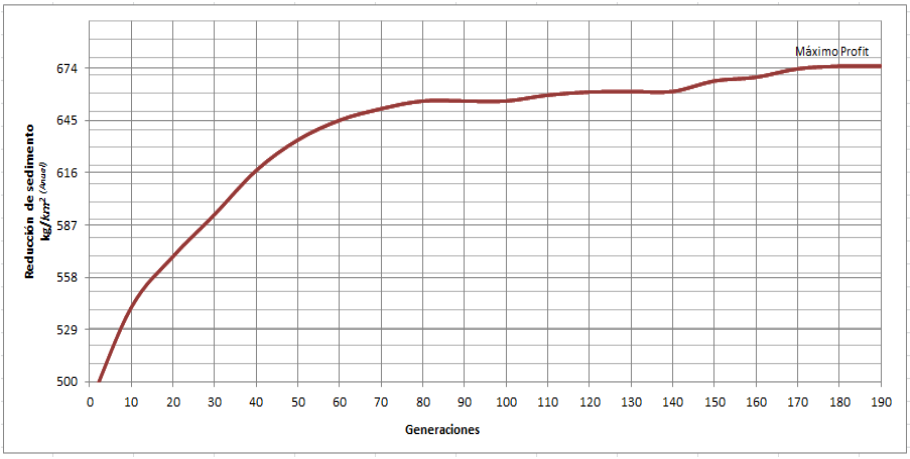


Figura 4.2: Con deformación del 10 % de la población

Figura 4.3: Algoritmo Genético, 2.000 celdas candidatas, Generaciones vs Beneficio

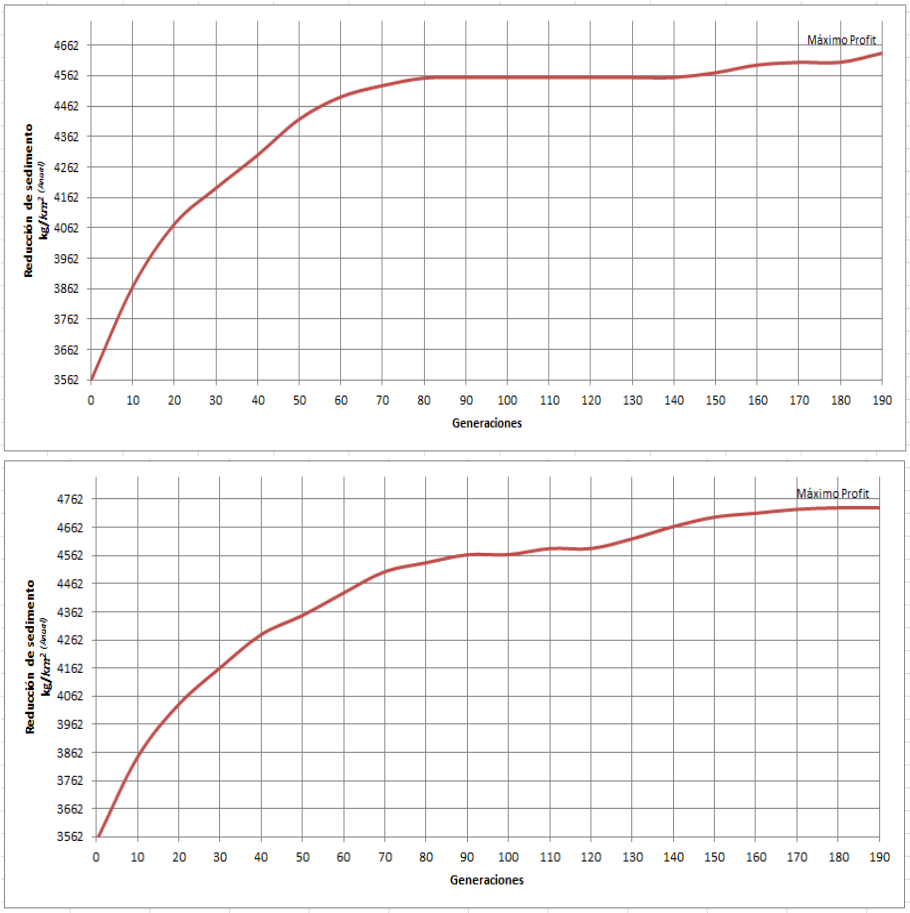


Figura 4.4: Con deformación del 10 % de la población

Figura 4.5: Algoritmo Genético, 3.000 celdas candidatas, Generaciones vs Beneficio

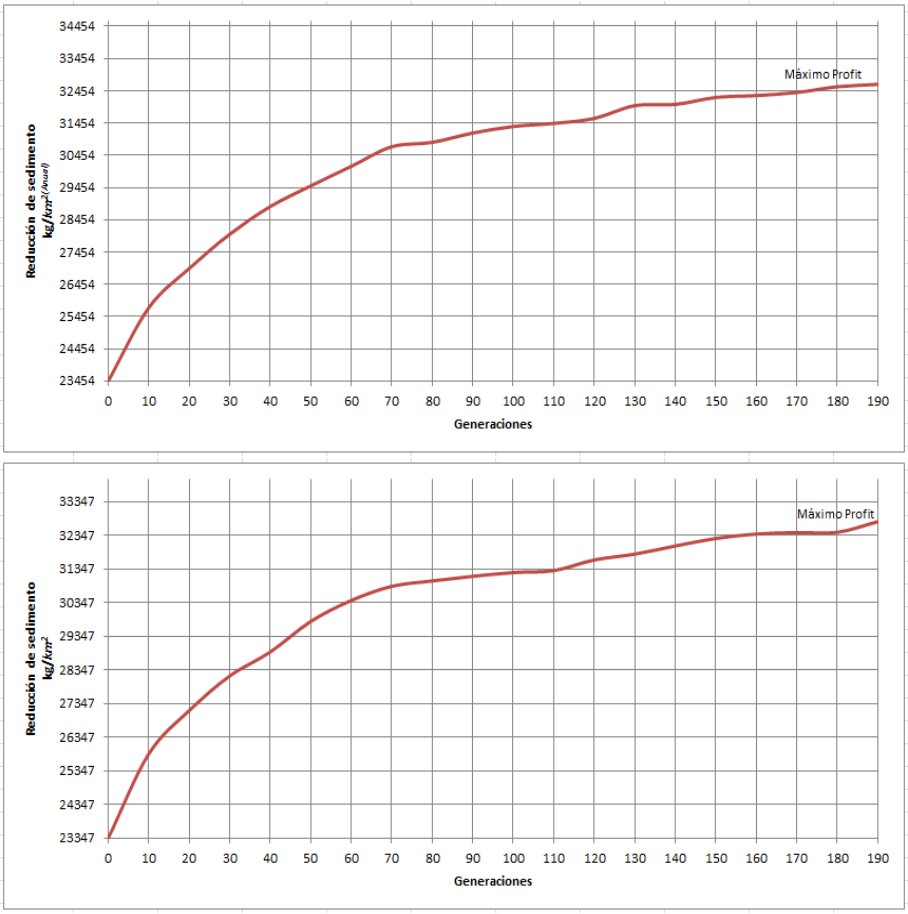


Figura 4.6: Con deformación del 10 % de la población

Figura 4.7: Algoritmo Genético, 4.000 celdas candidatas, Generaciones vs Beneficio

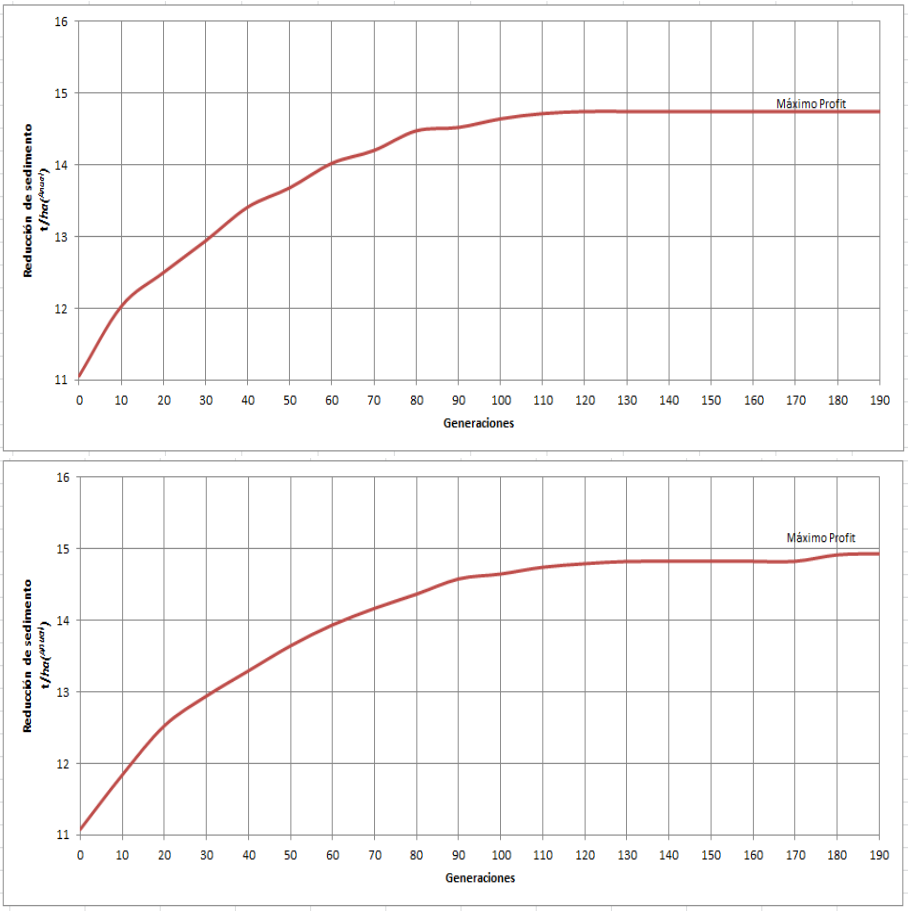


Figura 4.8: Con deformación del 10 % de la población

Figura 4.9: Algoritmo Genético, 15.014 celdas candidatas, Generaciones vs Beneficio

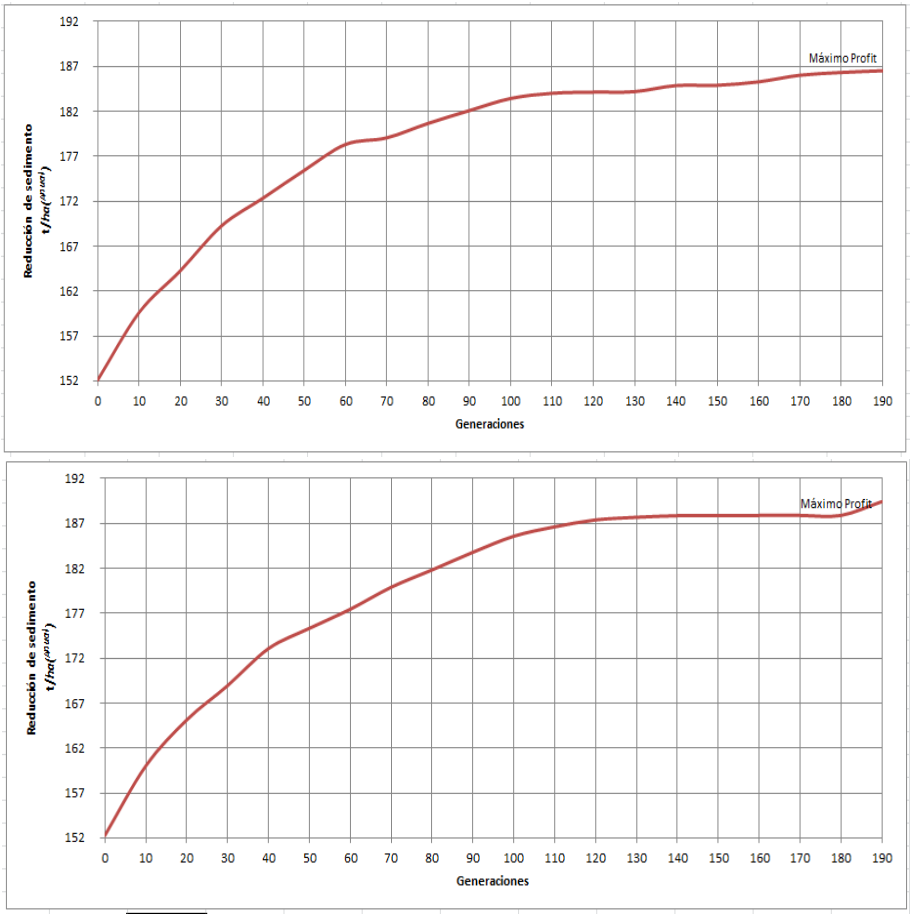


Figura 4.10: Con deformación del 10 % de la población

Conclusiones

Se ha modelado el problema de selección de áreas óptimas a ser forestadas, para minimizar la producción de sedimento, de una cuenca hidrográfica como un «Knapsack Problem» y se han probado nuevas estrategias con el *Algoritmo Genético*, *Branch and Bound*, y *deformación topológica*. El algoritmo *Branch and Bound* aplicado al problema de forestación modelado como un «Knapsack Problem» es una alternativa al algoritmo *Shani's PTAS algoritmo* de complejidad $O(n^{k+1})$ que se integró a *CAMF-B*, para resolver el problema de selección de áreas óptimas, con restricción presupuestaria, ya que nos permite calcular el óptimo global. *Branch and Bound* permite obtener soluciones óptimas para «Knapsacks» donde intervienen miles de celdas, en este proyecto los métodos propuestos se probaron en una parte de la cuenca del Tabacay correspondiente a 39.688 celdas, y en los problemas de forestación intervienen miles de celdas. En situaciones donde tenemos un número excesivo de celdas y no podemos aplicar el algoritmo *Branch and Bound*, el *Algoritmo Genético* ha demostrado ser una alternativa, aportando soluciones factibles, cercanas al óptimo global, lo que ya no se puede conseguir con métodos exactos. No es posible saber qué tan cercano referente al óptimo global se encuentra la solución para el problema donde intervienen 15,014 celdas al no tener un valor de referencia y al tratarse de un valor calculado con una heurística.

La variante del *algoritmo Genético* con *deformación topológica* como heurística, para generar buen material genético en la población inicial redundaron en mejoras mínimas del beneficio, que pueden seguir mejorando si se cuenta con más recursos que permitan disponer poblaciones más numerosas y más evoluciones.

El aporte de este proyecto es la integración de nuevos algoritmos, que permiten una eficiente selección de áreas óptimas para forestación, la aplicabilidad de los mismos a problemas reales, y la mejora en la selección usando *Branch and Bound* que se puede integrar al algoritmo CAMF-B.

Apéndice A

Código fuente del Algoritmo Genético en *java* con la librería *JGAP* versión 3.4.4

La librería *JGAP* especializada en Algoritmos Genéticos es software libre y se puede distribuir bajo la *GNU Lesser Public License* 2.1 o posterior. Las aplicaciones comerciales que no publiquen su código fuente deben distribuirse bajo *Mozilla Public License* para lo cual deben contribuir al proyecto con un aporte económico https://sourceforge.net/projects/jgap/files/.
JGAP dispone de clases e interfaces que permiten representar los genes, población, cromosomas y función de ajustes, donde se puede personalizar la función objetivo según el problema de optimización que tratamos de resolver. Además permite generar poblaciones de manera aleatoria y especificar parámetros del tamaño de la población, tipo de operador de cruzamiento, número de generaciones así como evoluciones, el algoritmo finalmente devuelve el individuo mejor adaptado con el método *getFittest-Chromosome()*.

A.1. Función objetivo

```
package algoritmogeneticooptimizado;
import org.jgap.FitnessFunction;
import org.jgap.IChromosome;
/**
 *
 * @author LuisEduardo
 */
    public class Funcion extends FitnessFunction {
        // variable presupuesto va ha corresponder equivalente al volumen del knapsack
        private final double presupuesto;
        //rho de penalización
        private double rho=0;
        // esta varible va a contener la sumatoria del costo total de forestación en el caso extremo que se forestan
        //todas las celdas
        private double presupuesstoMaximo;
        // vector con los costos de forestación expresados en unidades de celda
        private final double[] costo;
        // vector con los valores correspondientes a la reduccion de sedimentos a la salida de la cuenca
        private final double[] sedimento;
        public Funcion(double presupuesto, double[] costos, double[] sedimentos) {
                if( presupuesto < 0 ) throw new IllegalArgumentException("El Knapsack debe tener un"
                        + " presupuesto mayor a cero");
                this.presupuesto = presupuesto;
                int i = 0;
                costo = new double[costos.length];
                for(i=0, presupuesstoMaximo = 0;i<costos.length;i++) {
                        costo[i]=costos[i];
                        presupuesstoMaximo += costos[i];

                }
                sedimento = new double[sedimentos.length];
                for(i=0;i<sedimentos.length;i++){
                    sedimento[i]=sedimentos[i];
                    if(i==0){rho=sedimento[i]/costo[i];}
                    else if(rho<(sedimento[i]/costo[i]))
                    {
                        rho=sedimento[i]/costo[i];
                    }
                }
        }
/* Sobreescribimos la funcion de evaluación, para asignar el fitness comprobamos que no exceda el presupuesto,
   si este sobrepasa el presupuesto fijo, el cromosoma es penalizado de forma lineal, esto evitara las soluciones
   no factibles, para aplicar el operador de reparacion tambien se podria modificar en esta bloque en lugar de
        penalizar al cromosoma se podria reparar este cromosoam con la funcion reparación.
*/        @Override
        protected double evaluate(IChromosome ch) {
                //double eval = presupuesstoMaximo*presupuesstoMaximo;
                double eval =0;
                double costoCromosoma = 0;
                for(int i = 0 ; i < costo.length ; i++){
                        eval += getCeldaCeleccionada(ch, i)*sedimento[i];
                        costoCromosoma += getCeldaCeleccionada(ch, i)*costo[i];
                }
                if(costoCromosoma>presupuesto) {
                        //eval -= getRelacionSedimentoCosto()*Math.log(1+costoCromosoma); // log
                        eval -= rho*(costoCromosoma-presupuesto); // penalización linear
                        costoCromosoma=0;
                }
                else return eval;
                if(eval<0)return 0;
```

```
                else return eval;
        }
        public int getCeldaCeleccionada(IChromosome chr, int i) {
                if ((Boolean)chr.getGene(i).getAllele()) return 1;
                else return 0;
        }
        public double getTotalCosto(IChromosome chr){
                double totalCosto = 0;
                for(int i = 0 ; i < costo.length ; i++){
                        totalCosto += getCeldaCeleccionada(chr, i)*costo[i];
                }
                return totalCosto;
        }
        public double getTotalBeneficio(IChromosome chr){
                double totalBeneficio = 0;
                for(int i = 0 ; i < costo.length ; i++){
                        totalBeneficio += getCeldaCeleccionada(chr, i)*sedimento[i];
                }
                return totalBeneficio;
        }

}
```

A.2. Función main

```
package algoritmogeneticooptimizado;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.PrintStream;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.jgap.Chromosome;
import org.jgap.Configuration;
import org.jgap.FitnessFunction;
import org.jgap.Gene;
import org.jgap.Genotype;
import org.jgap.IChromosome;
import org.jgap.Population;
import org.jgap.impl.BooleanGene;
import org.jgap.impl.DefaultConfiguration;
import org.jgap.impl.MutationOperator;

/**
 *
 * @author LuisEduardo
 */
public class AlgoritmoGeneticoOptimizado {

    public static final String SEPARATOR = "\t";
    public static final String QUOTE = "\"";
    static int numeroDeGenes = 1000;
    static double Presupuesto = 0.0;
    static double[] costosDeForestacion = new double[numeroDeGenes];
    int i = 0;
    static double[] beneficios = new double[numeroDeGenes];
    static FileOutputStream os;

    private static void cargarDatos(String nombreArchivo) {
        File file = new File(nombreArchivo);
        try {

            String objetosConPesos;
            FileReader f = new FileReader(file);
            BufferedReader Buffer = new BufferedReader(f);

            objetosConPesos = Buffer.readLine().trim();
            String[] MATRIZ = objetosConPesos.split(SEPARATOR);
            Presupuesto = Double.valueOf(MATRIZ[1]);
            objetosConPesos = Buffer.readLine();

            int contador = -1;

            while ((objetosConPesos = Buffer.readLine()) != null) {
                contador++;
                //separamso de la matriz no datos
                MATRIZ = objetosConPesos.split(SEPARATOR);
                costosDeForestacion[contador] = Double.valueOf(MATRIZ[0]);
                beneficios[contador] = Double.valueOf(MATRIZ[1]);
            }
            Buffer.close();
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
            System.out.println("Revisar Formato");
        }

    }

    /**
     * Evoluciones permitidas
     */
    private static final int evoluciones = 200;

    /**
     * *
     * función para calcular el costo maximo de forestación
     */
    public static void imprimirResultados(int generacion, double profit, String nombre) {

        FileOutputStream os;

        try {
            os = new FileOutputStream(nombre);
            PrintStream ps = new PrintStream(os);
            int contador = 0;

            ps.println(profit + "\t" + generacion);

        } catch (FileNotFoundException ex) {
            Logger.getLogger(AlgoritmoGeneticoOptimizado.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

    public static void costoMaximo() {
        double costoTotal = 0;
        for (int i = 0; i < beneficios.length; i++) {
            costoTotal += costosDeForestacion[i];
        }
        System.out.println("Costo total: " + costoTotal);
    }

    /**
     * Función que modifica la estructura topologíca de los cromosomas no
     * factibles esta deformación es una transoformación continua de la
     * estructura topologíca del cromosomas
     *
     */
    public static void deformarTopologia(List chromosomes, int porcentaje) {

        for (int i = 0; i < porcentaje; i++) {
            Object chromosome = chromosomes.get(i);
            IChromosome chrom = (IChromosome) chromosome;
            for (int j = 0; j < chrom.size(); j++) {
                Gene gene = chrom.getGene(j);
                gene.setAllele(true);
                //System.out.println("Gen : " + gene.getAllele().toString());
            }

        }
        Random ri = new Random();
```

```
        for (int i = 0; i < porcentaje; i++) {
            Object chromosome = chromosomes.get(i);
            IChromosome chrom = (IChromosome) chromosome;
            int deformaciones = ri.nextInt(chrom.size());
            for (int j = 0; j < deformaciones; j++) {
                int indiceDeformacion = ri.nextInt(chrom.size());
                Gene gene = chrom.getGene(indiceDeformacion);
                gene.setAllele(false);
                //System.out.println("Gen : " + gene.getAllele().toString());
            }

        }
    }

    public static void main(String[] args) {
        //cargamos los datos del el Knapsack
        cargarDatos("1000.knp");
        //costoMaximo();
        double PresupuestoMaximo = Presupuesto;
        // Usamos una configuracion predeterminada
        Configuration conf = new DefaultConfiguration();
        // varible donde guradamos la población tipo Genotype
        Genotype population = null;
        // Definimos una función para calcular el fitness de los cromosomas
        FitnessFunction myF = null;

        try {
            /* Instancía la funcion de fitenes necesitamos el presupuesto del  proyecto,
                vectores de costoDeForestacion y los beneficios cargados en memoria.
             */
            myF = new Funcion(PresupuestoMaximo, costosDeForestacion, beneficios);

            conf.setFitnessFunction(myF);


            /* Definimos el tipo de cromosoma y la longitud(tamaño de los knapsacks)
                        * en esta parte definimos de tipo booleano cada gen
             */
            Gene[] genes = new Gene[beneficios.length];
            double volumen = 0;
            for (int i = 0; i < beneficios.length; i++) {
                genes[i] = new BooleanGene(conf);
            }

            IChromosome cromosomas = new Chromosome(conf, genes);

            conf.setSampleChromosome(cromosomas);
            // definimos tamaño de la poblacion incial
            conf.setPopulationSize(300);
            //public static MutationOperator mutacion=new MutationOperator(this,0.05);
            //mutacion.setMutationRate(0.05);
            //conf.addGeneticOperator(mutacion);
            // despues de generar la descendencia porcentaje que se debe conservar
            conf.setSelectFromPrevGen(1);

            conf.setPreservFittestIndividual(true);// se almacene el valor más óptimo hasta la fecha
            //population = Genotype.;
            // definimos población inicial
            population = Genotype.randomInitialGenotype(conf);
```

```
            Population p = population.getPopulation();
            List chromosomes = p.getChromosomes();

            //apicamos el operador de defomacion segun se requiera
           //deformarTopologia(chromosomes, 30);
            /*
    for (Object chromosome : chromosomes) {
     IChromosome chrom = (IChromosome) chromosome;
     for (int j = 0; j < chrom.size(); j++) {
         Gene gene = chrom.getGene(j);
        // gene.setAllele(true);
        System.out.println("Gen : " + gene.getAllele().toString());
     }
}*/         } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(0);
        }

        //
        IChromosome best = null;

        try {
            os = new FileOutputStream("knapsack1000.kp");
        } catch (FileNotFoundException ex) {
            Logger.getLogger(AlgoritmoGeneticoOptimizado.class.getName()).log(Level.SEVERE, null, ex);
        }

        for (int i = 0; i < evoluciones; i++) {

            population.evolve();
            if (i % 10 == 0) {
                    // cada 10 evoluciones escribimos el mejor ADN
                best = population.getFittestChromosome();

                PrintStream ps = new PrintStream(os);
                int contador = 0;

                ps.println(((Funcion) myF).getTotalBeneficio(best) + "\t" + i);

            }

        }

        best = population.getFittestChromosome();
        // Valor óptimo Final
        double profit = 0;
        double volumen = 0;
        System.out.println("Costo: " + ((Funcion) myF).getTotalCosto(best));
        System.out.println("Beneficio: " + ((Funcion) myF).getTotalBeneficio(best));
        for (int j = 0; j < beneficios.length; j++) {
            if ((Boolean) best.getGene(j).getAllele()) {
                //System.out.println(beneficios[j]);
                profit += beneficios[j];
                volumen += costosDeForestacion[j];
            }
        }
        System.out.println("El valor maximo es" + profit + "con volumen" + volumen);

    }
}
```

Apéndice B

Código fuente del algoritmo *Branch and Bound* en *java*

B.1. Clase Cota

El objetivo de esta clase es almacenar en cada celda si fué elegida para ser forestada.

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package mochila;

/**
 *
 * @author LuisEduardo
 */
// acumulamos las limintes superiores para llevar todas las restricciones aqui
public class Cota {
        public  int indices;
        //bandera para saber si la celda es forestada
        public boolean forestada;

        public Cota() {
        }

    public Cota(int indices, boolean forestada) {
        this.indices = indices;
        this.forestada = forestada;
    }


    public int getIndices() {
        return indices;
    }

    public void setIndices(int indices) {
        this.indices = indices;
    }

    public boolean isForestada() {
        return forestada;
    }

    public void setForestada(boolean forestada) {
        this.forestada = forestada;
    }



}
```

B.2. Clase objetos

Esta clase representa a las celdas, con los parámetros de costo de forestación y beneficio asociado.

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package mochila;

/**
 *
 * @author LuisEduardo
 */
class Objetos implements Comparable<Objetos> {
        int codigo;
        double costo;
        double beneficio;

        public Objetos(int codigo,double costo,double beneficio){
            this.codigo = codigo;
            this.costo=costo;
            this.beneficio=beneficio;
        }

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public double getcosto() {
        return costo;
    }

    public void setcosto(double costo) {
        this.costo = costo;
    }

    public double getbeneficio() {
        return beneficio;
    }

    public void setbeneficio(double beneficio) {
        this.beneficio = beneficio;
    }

            @Override
        public int compareTo(Objetos objetos) {
            //comparamos la costo del objeto actual con siguiente objeto para ordenar
            if( (this.beneficio/this.costo)>(objetos.beneficio/objetos.costo) ) return 1;
            else if( (this.beneficio/this.costo)<(objetos.beneficio/objetos.costo) ) return -1;
            else return 0;
        }


}
```

B.3. Función main

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package mochila;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
import java.util.Stack;
import java.util.logging.Level;
import java.util.logging.Logger;
import sun.reflect.generics.tree.Tree;

/**
 *
 * @author LuisEduardo
 */
public class Mochila implements Tree{
    //variables globales
    List<Objetos> objetos=new ArrayList();
    List<Objetos> objetosAux=new ArrayList();
    double costoTotal;
    public static final String SEPARATOR = "\t";
    public static final String QUOTE = "\"";
    //limite inferior
    double LB=0;
    double[] optimo;
     static String optimoGloval=new String();


    private void cargarDatos(String nombreArchivo){
        File file = new File(nombreArchivo);
        try{

            String objetosConPesos;
            FileReader f = new FileReader(file);
            BufferedReader Buffer = new BufferedReader(f);

            objetosConPesos = Buffer.readLine().trim();
            String[] MATRIZ = objetosConPesos.split(SEPARATOR);
            costoTotal = Double.valueOf(MATRIZ[1]);
            objetosConPesos = Buffer.readLine();

            int contador=0;

            while((objetosConPesos = Buffer.readLine())!=null) {
                 contador++;
```

```
                //separar matriz de datos no validos
                MATRIZ = objetosConPesos.split(SEPARATOR);
                objetos.add(new Objetos(contador,Double.valueOf(MATRIZ[0]),Double.valueOf(MATRIZ[1])));
            }
            Buffer.close();
            Collections.sort(objetos);
            Collections.reverse(objetos);
        }
        catch(Exception ex){
            ex.printStackTrace();
            System.out.println("Revisar Formato");
        }

    }



    public Mochila(String nombre,Cota []beta){

        cargarDatos(nombre);
        abrirNodos(beta);

    }



    public void abrirNodos(Cota[] cotas){
```

```
        double volumen=this.costoTotal;
        double[] currentSolucion = new double[this.objetos.size()];
        for(Cota viaje:cotas){
            if(viaje.forestada){
                double costoObjeto = objetos.get(viaje.indices).costo;
                if( costoObjeto<= volumen){ // probamos si puede entrar en a mochila
                    volumen-=costoObjeto;
                    currentSolucion[viaje.indices]=1.0;
                   //Bien entra en la mochila
                }
                else{
                    return;
                    //Solucion con decimales
                }
            }
        }

        for(int i=0;i<this.objetos.size();i++){
// verificamo par el conjunto de los nodos restantes de las restricciones
// cuando hay conflicto
            boolean conflicto=false;
            for(Cota viaje:cotas){
                if(viaje.indices==i){
                    conflicto=true;
                    break;
                }
            }
            if(conflicto) continue;
```

```
                double volumeElemento = objetos.get(i).costo;
                if( volumeElemento<= volumen){ //verificamos
                    volumen-=volumeElemento;
                    currentSolucion[i]=1.0;
                    //System.out.println("Solucines"+currentSolucion[i]);
                }
                else{
                    currentSolucion[i]=volumen/volumeElemento;
                    double limitePrueba=limiteInferior(currentSolucion);
                    if(limitePrueba<this.LB) return;
                //podamos el arbol hijos con soluciones malas

//generamos nuevas restricciones
                    Cota[] forestada = new Cota[cotas.length+1];
                    Cota[] noforestada = new Cota[cotas.length+1];

            //heredamos restricciones
                    for(int j=0;j<cotas.length;j++){
                        forestada[j]=cotas[j];
                        noforestada[j]=cotas[j];
                    }
                    forestada[forestada.length-1]=new Cota(i,false);
                    noforestada[noforestada.length-1]=new Cota(i,true);
                    abrirNodos(forestada);
                    abrirNodos(noforestada);
                    return;
                }
            }
            for(int i=0;i<objetosQueforestada.length;i++){
                if(objetosQueforestada[i]==1.0) {
                    maximizacion+=objetos.get(i).beneficio;
                }

            }
            return maximizacion;
        }

        public double limiteInferior(double[] beta){
            double limiteInferior=0;
            for(int k=0;k<beta.length;k++){
                limiteInferior+=(beta[k]*objetos.get(k).beneficio);
            }
            return limiteInferior;
        }


        public static void main(String[] args) {
            // TODO code application logic here
            Cota []alfa=new Cota[0];
            Mochila mochila=new Mochila("4000.knp",alfa);
            double valor=mochila.LB;
            System.out.println("\n optimo global\n");
            System.out.println("Optimo Global: "+valor);

        }
```

Apéndice C

Diagrama del Algoritmo Genético por componentes

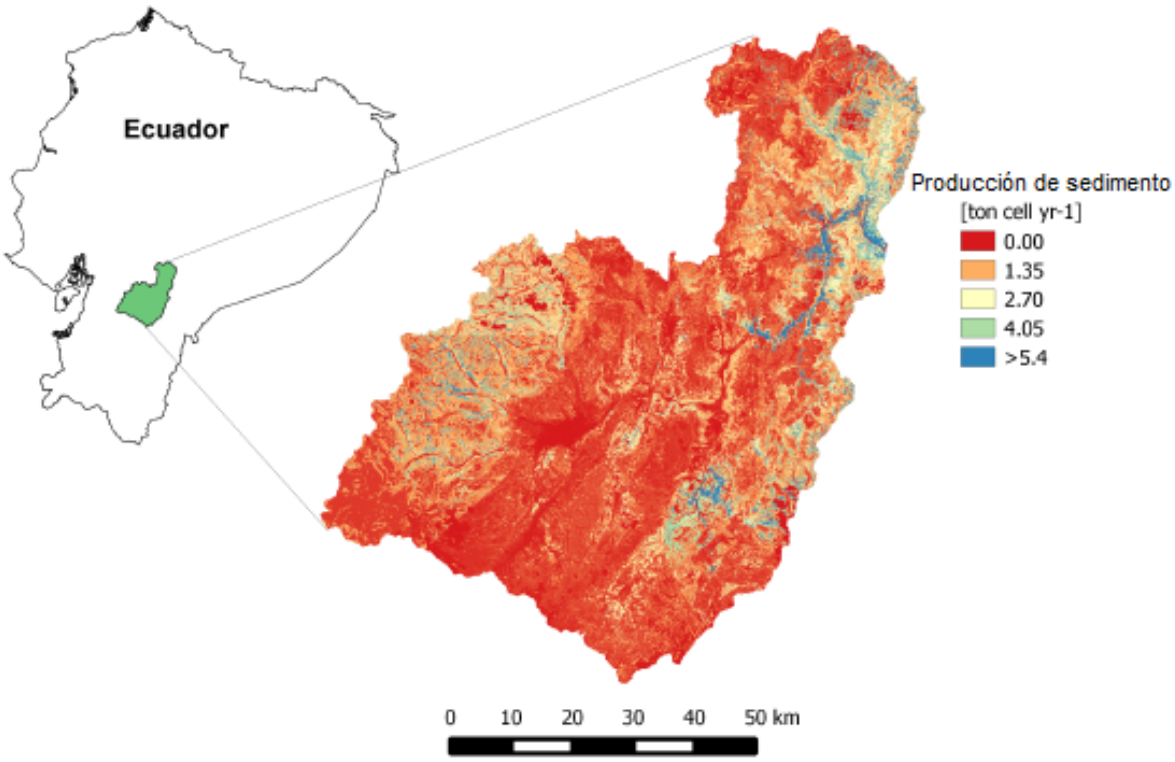

FONS
VITÆ
CREDITI
POSIDENTIS
UNIVERSIDAD DE CUENCA

C.1. Componentes del Algoritmo Genético

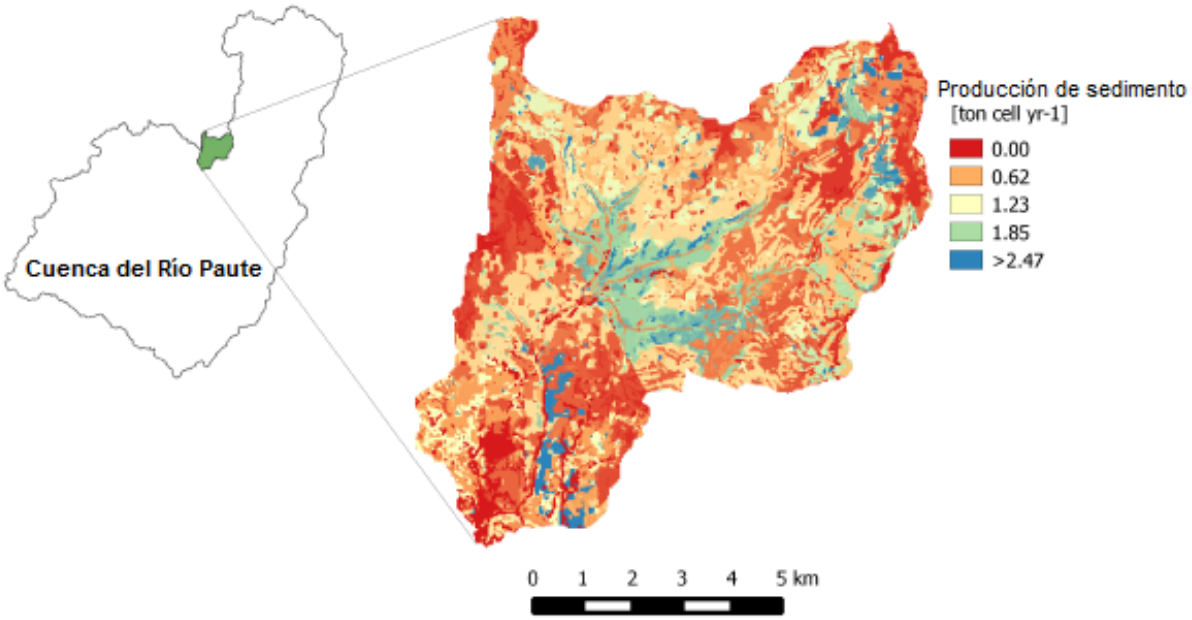
Apéndice D

Mapas de la región de estudio

D.1. Mapas Cuenca del Paute



D.2. Mapas de la cuenca del Río Tabacay



Referencias

Affenzeller, M., Wagner, S., Winkler, S., y Beham, A. (2009). *Genetic algorithms and genetic programming: modern concepts and practical applications*. Chapman and Hall/CRC.

DeJong, K. (1975). Analysis of the behavior of a class of genetic adaptive systems. dept. *Computer and Communication Sciences, University of Michigan, Ann Arbor*.

Estrella, R. (2015). Where to afforest? single and multiple criteria evaluation methods for spatio-temporal decision support, with application to afforestation.

Estrella, R., Vanegas, P., Cattrysse, D., y Van Orshoven, J. (2014). Trading off accuracy and computational efficiency of an afforestation site location method for minimizing sediment yield in a river catchment. En *Proceedings of GEOProcessing 2014: The Sixth International Conference on Advanced Geographic Information Systems, Applications, and Services*, pages 94–100. International Academy, Research, and Industry Association (IARIA).

FAO (1979). *A provisional methodology for soil degradation assessment*.

Fiume, E. L. (2014). *The mathematical structure of raster graphics*. Academic Press.

Goldberg, D. E. (2002). *Genetic algorithms*. Pearson Education, Singapore.

H. Keith Moffatt (auth.), R. L. R. e. (2001). *An Introduction to the Geometry and Topology of Fluid Flows*. NATO Science Series 47 Series II. Springer Netherlands, 1 edition.

Jenson, S. K. y Domingue, J. O. (1988). Extracting topographic structure from digital elevation data for geographic information system analysis. *Photogrammetric engineering and remote sensing*, 54(11):1593–1600.

Karp, R. M. (1972). Reducibility among combinatorial problems. En *Complexity of computer computations*, pages 85–103. Springer.

Kellerer, H., Pferschy, U., y Pisinger, D. (2004). Introduction to np-completeness of knapsack problems. En *Knapsack problems*, pages 483–493. Springer.

Kolesar, P. J. (1967). A branch and bound algorithm for the knapsack problem. *Management science*, 13(9):723–735.

Luke, S. (2013). *Essentials of Metaheuristics*. Second edition edition.

Mansfield, M. (1964). Introduction to topology. En *The University Series in undergraduate mathematics*, page 15. D VAN NOSTRAND COMPANY, INC PRINCENTON, NEW JERSEY.

Martello, S. (1990). Knapsack problems: algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimiza tion*.

Michalewicz, Z. y Hartley, S. J. (1996). Genetic algorithms+ data structures= evolution programs. *Mathematical Intelligencer*, 18(3):71.

Molina, A., Govers, G., Poesen, J., Van Hemelryck, H., De Bièvre, B., y Vanacker, V. (2008). Environmental factors controlling spatial variation in sediment yield in a central andean mountain area. *Geomorphology*, 98(3-4):176–186.

Palmieri, A., Shah, F., y Dinar, A. (2001). Economics of reservoir sedimentation and sustainable management of dams. *Journal of environmental management*, 61(2):149–163.

Polya, G. (2014). *How to solve it: A new aspect of mathematical method*. Princeton university press.

Rajasekaran, S. y Pai, G. V. (2003). *Neural networks, fuzzy logic and genetic algorithm: synthesis and applications (with cd)*. PHI Learning Pvt. Ltd.

Rashid, A. B. (2010). *The 0-1 Knapsack Problem A solution by Genetic Algorithm*. VDM Verlag Müller.

Sahni, S. (1975). Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)*, 22(1):115–124.

Sánchez, J. M. y Pérez (2013). Caracterización y análisis de los sistemas de terrazas agrícolas en el valle de toluca, méxico. *Agricultura, sociedad y desarrollo*, 10(4):397–418.

Satō, H. (1999). *Algebraic topology: an intuitive approach*. American Mathematical Soc.

Tetsuya Kusuda, Hiroyuki Yamanishi, J. S. J. Z. G. (2007). *Sediment and Ecohydraulics, Volume 9: INTERCOH 2005*. Proceedings in Marine Science 9. Elsevier.

Threlfall., H. S. Y. W. (1951). *Lecciones de Topología.* Primera editio edition.

Tobler, W. R. (1979). Cellular geography. En *Philosophy in geography*, pages 379–386. Springer.

Tokieda, T. (2001). Topology in four days. En *An Introduction to the Geometry and Topology of Fluid Flows*, pages 35–55. Springer.

Vanegas, P., Cattrysse, D., y Van Orshoven, J. (2010). Budget constraint in reforestation meant for minimizing sediment load at a watershed outlet.

Vanegas Peralta, P. F., Cattrysse, D., y Van Orshoven, J. (2010). Sediment flow minimization requirements in reforestation initiatives: a heuristic solution method. *Computers & Operations Research*.

Vladimir I. Arnol'd, Vladimir I. Arnold, R. C. (1992). *Ordinary differential equations*. New York :, Springer-Verlag, [3rd ed.] edition.

Von Neumann, J., Burks, A. W., et al. (1966). Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14.

Winston, W. L. y Goldberg, J. B. (2005). Investigación de operaciones: aplicaciones y algoritmos.

Wolock, D. M. y McCabe Jr, G. J. (1995). Comparison of single and multiple flow direction algorithms for computing topographic parameters in topmodel. *Water Resources Research*, 31(5):1315–1324.

Yoo, J., Simonit, S., Connors, J. P., Kinzig, A. P., y Perrings, C. (2014). The valuation of off-site ecosystem service flows: Deforestation, erosion and the amenity value of lakes in prescott, arizona. *Ecological Economics*, 97:74–83.